EXPLORATION OF DIGITAL CIRCUITS AND TRANSISTOR-LEVEL TESTING IN THE DARPA TRUST PROGRAM

THESIS

Ralph K. Tatum, Second Lieutenant, USAF

AFIT-ENG-MS-15-M-040

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-15-M-040

EXPLORATION OF DIGITAL CIRCUITS AND TRANSISTOR-LEVEL TESTING IN THE DARPA TRUST PROGRAM

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Ralph K. Tatum, BS

Second Lieutenant, USAF

March 2015

AFIT-ENG-MS-15-M-040

EXPLORATION OF DIGITAL CIRCUITS AND TRANSISTOR-LEVEL TESTING IN THE DARPA TRUST PROGRAM

THESIS

Ralph K. Tatum, BS

Second Lieutenant, USAF

Committee Membership:

Maj Derrick Langley, PhD
Chair

Mary Y. Lanzerotti, PhD
Member

Kenneth M. Hopkinson, PhD
Member

Maj Samuel Stone, PhD
Member

Abstract

The need to verify correct circuit operation has grown in recent years due to adversaries ability to compromise DoD systems. The DARPA program addressed this issue and implemented the DARPA TRUST program to verify untrusted circuits using software. The DARPA TRUST program was initiated in 2006 and due to this the limitations and potential errors in the program have not yet been fully explored.

This research identifies the potential errors in the program by conducting transistor-level testing on circuits. The DARPA TRUST program currently operates at the gate-level and conducting various experiments at the transistor-level brought to light potential problems with current DARPA TRUST testing. The way that transistor-level verification is conducted is through netlist matching. A schematic of a circuit is created and the netlist is extracted, after that a metal layout of a circuit is created and the netlist is extracted. Once the two netlists are extracted, a matching program is used and the result determines if the verification process is successful. Parasitic capacitance was extracted in the metal layout version of a circuit and netlists were compared with the schematic version. Results show that parasitic capacitance is overlooked in the DARPA TRUST program even though this could potentially cause a fabricated device to fail. Transmission lines were simulated by creating metal wiring between two inverters. These metal lines mimic the operation of a transmission line. These transmission lines were experimented on and it was determined that the DARPA TRUST program does not effectively check for potential errors in transmission line fabrication. The results of this research brought to light the vulnerabilities in the DARPA TRUST program and addressed the need for the program to conduct transistor-level testing.

Table of Contents

Page

List of Figures

List of Tables

List of Symbols

Symbol	Definition
Subscripts	
0	center
L	low
H	high

List of Acronyms

Acronym	Definition
AFRL	Air Force Research Laboratory
FIR	finite impulse response
FFT	Fast-Fourier transform
DTICS	Defense Trusted Integration Circuit Strategy
TF	Trusted Foundry
DARPA	Defense Advanced Research Projects Agency
DMEA	Defense Microelectronics Activity
IP	intellectual property
IC	integrated circuits
RTL	register-transfer level
MSDC	Mixed Signals Design Center
AFIT	Air Force Institute of Technology
NCSU	North Carolina State University
DoD	Department of Defense
PDK	process design kit
LVS	layout vs. schematic
VLSI	Very-large-scale integration
MIPS	microprocessor without interlocked pipeline stages
MTO	Microsystems Technology Office
AES	Advanced Encryption Standard
TRUST	Trusted for Integrated Circuits
EDA	electronic design automation

EXPLORATION OF DIGITAL CIRCUITS AND TRANSISTOR-LEVEL TESTING IN THE DARPA TRUST PROGRAM

I. Introduction

In 2006, the Defense Advanced Research Projects Agency (DARPA) established the Trusted for Integrated Circuits (TRUST) program to verify circuit operation manufactured from untrusted sources due to the Department of Defense (DoD) need for trustworthy hardware [6]. In recent years, the necessity to provide accurate testing on integrated circuits has increased. In 2008 Business Week published an article bringing to light threats on integrated circuits due to companies outsourcing their products [9]. This article discussed the potential for compromised designs due to companies outsourcing the fabrication of their circuits. In particular, four counterfeit Xicor chips were discovered in the flight computer of an F-15 fighter jet at Warner Robins Air Force Base [9]. Another example of circuit tampering happened in 2007 in the Syrian military. A Syrian radar failed to warn of an incoming air strike and a backdoor built into the system's chips were rumored to be responsible [29]. Due to the necessity of verifying correct circuit operation, the DoD wants to establish a reliable supply of custom hardware [23]. Very-large-scale integration (VLSI) circuits designed for the DoD are typically low-volume products that are not highly profitable for commercial manufacturers [7]. However, the Department of Defense also relies on commercial built hardware in their systems that are mass produced and the need to verify the correct operations of these circuits is a necessity [28]. Additionally, custom hardware in DoD systems have a strict set of specifications that extend beyond commercial chip requirements for environmental factors, reliability, and useful life [26]. The supply

chain for these custom DoD hardware systems must provide functional, trusted hardware as well as being competitive with available commercial technologies [20].

1.1 DARPA TRUST

The United States does not have a comprehensive program to certify that integrated circuits (IC) going into U.S. weapon systems do not contain malicious circuits [14]. Due to these concerns, DARPA initiated the TRUST program to develop a system to ensure the trust of ICs used in military infrastructure, but designed and fabricated under untrusted conditions. Untrusted conditions means any point in the process of circuit inception to fabrication that has the potential to produce a bad or tampered integrated circuit [25].

Differing from other approaches to verifying circuit operation, the DARPA TRUST program assigns measurable metrics to determine correct circuit operation [14]. These two metrics are P_D and P_{FA} which corresponds to the probability of detecting a malicious transistor and the probability of falsely identifying an operational transistor as malicious. The measurable metrics and testing method used by the DARPA TRUST program have never been used in circuit verification before, which makes this program a unique view on trusting circuits. These two metrics are; the probability of detecting a malicious insertion and the probability of false alarms. A more traditional approach to the identification of a circuit that was maliciously attacked is to have the Trojan Horse being the signal. The term Trojan Horse was originally invented when the Greek soldiers tried to invade Troy but were unsuccessful. They ultimately left but provided a wooden horse as a gift at the gate of Troy. This horse was taken into the city and during the night Greek soldiers came out of the horse and destroyed the city. Nowadays, a Trojan Horse is a seemingly innocuous piece of hardware or software that actually is malicious to the design [29]. However, the TRUST program uses a more basic measurement where any change in the IC is considered to be a malicious attack. The first DARPA TRUST experiment was testing and verifying a 64-bit adder that had two malicious transistors added to the design [8].

1.2 Research Problem

This research focuses on the use of digital circuits in integrated circuit verification. Since the DARPA TRUST program is fairly new, the research limitations of the program has not been fully explored. This research will further explore the limitations to see if real world digital circuits can be verified with the DARPA TRUST tools as well as possible problems with the current DARPA TRUST program. In recent years, there has been a movement from analog circuits to digital circuits in everyday use, and therefore the need to verify these digital circuits is substantial. One of the reasons for this is that computers are mostly digital because they run on a clock. Also, digital circuits enables smaller size, lower power, and lower parts counts [27].

Digital circuits are made up of combinatorial logic gates connected to a clock to synchronize events between these gates. The number of logic gates between each digital circuit provide a range based on technology and to this date there has not been a characteristic study on the amount of logic gates versus the TRUST program output. For example, a basic XOR2 gate is comprised of 5 NAND2 gates, and a 1-bit full adder has 13 NAND2 gates. Usually adders are not one bit and for a simple 16-bit adder the number of NAND2 gates is 208 NAND2 gates. As one can see, the number of gates scales between just these two simple circuits. In this research, digital circuits will be pulled from a third party source (opencores.org), and run through the DARPA TRUST suite to determine each circuit's metrics. Each digital circuit will be optimized under different parameters (power, speed, area, etc.) to see if there is any difference in the output of the DARPA TRUST program. Once these results have been achieved, a better understanding on the correlation between the number of logic gates and the DARPA TRUST output will be achieved. Also, the results could potentially show which digital structures are more vulnerable to tampering. For example, an AND gate might be more likely to be tampered with than an inverter gate.

Various circuit changes will be implemented to see if the DARPA TRUST program can handle these changes. These circuit changes will include custom vs. built-in circuits taken from a library, the effect of parasitic capacitance on circuits, and if transmission lines affects circuit operation. As of right now, the DARPA TRUST program only operates at the gate-level. Making these small changes in circuits will affect the transistor-level of circuits. Hopefully this research can try to exploit possible vulnerabilities and could prove that the TRUST program should also look at doing transistor level testing of circuits.

1.3 Justification

As stated in the article presented by [9], the DoD has a need to verify circuits fabricated from untrusted sources. The potential for our adversaries to compromise our hardware components for espionage is far too great for the DoD to idly stand by. This is why the DARPA TRUST program was initiated and continues to be a vital component in the war fighting effort of the DoD. One reason that the DARPA TRUST program exists is that fabrication centers are very expensive to build in today's economy. Even though the US has microelectronic fabrication centers, a lot of the time the DoD outsources the majority of the fabrication process to outside centers since it is cheaper [17]. Since the DoD does not have it's own fabrication center, it is even more paramount that digital circuits need to be verified since it's source could potentially be corrupted.

Not all malicious attacks occur at the beginning of circuit operation. Sometimes these attacks occur slowly over time like a ticking time bomb that is just waiting for the right time to go off. These tampered with parts can be disguised as military grade electronics that are just waiting for the opportune time to cause havoc on DoD systems. These parts may not fail until subjected to environmental stresses outside the normal, commercial specification [17]. Therefore, it is a necessity that the DARPA TRUST program exists to counteract these potential security threats dealing with integrated circuits.

1.4 Proposed Methodology

This research builds upon previous algorithms implemented in software by contractors in pursuit of the DARPA TRUST program. In particular, the research follows on with research conducted by 2Lt Michael Seery [28]. This research will build on the existing DARPA TRUST output toolset by investigating success and failure cases of digital circuits, differing in logic gate complexity, and attempting to push the DARPA TRUST program to its limit at the transistor-level.

One of the first steps in this research is to identify which intellectual property (IP) cores to test with the DARPA TRUST program. First, a baseline experiment will be conducted to provide basic outputs and then different IP cores will be selected based on the number of logic gates to increase the complexity of the circuit. These cores will be synthesized in software under different constraints (speed, power, area, current levels, bias conditions, etc.), to enable the utilization of various and wider logic gates. The synthesis of these cores will be performed in Cadence Encounter software, and the verification will be performed in Cadence Conformal. The verification process deals with the mapping of netlist created both in the forward and reverse direction of the digital circuit creation process. This process will help identify how the different constraints affect the DARPA TRUST tools throughput given various synthesized known circuit cores. The same circuit that is optimized for speed might have a different output from the DARPA TRUST tools than that same circuit optimized for area. The goal is to identify how the different constraints affect the performance of the DARPA TRUST tool in detecting malicious contents.

Another area that this research will focus on is varying circuit inception to see if there are any problems in the DARPA TRUST program. For example, when fabricating a circuit if built-in components are used, the custom component originally created might be different than the built-in component. If this is the case, then the DARPA TRUST output

might produce a bad result when in reality it is correct. Also, by extracting the parasitic capacitance in the metal layout version of a digital component the DARPA TRUST output could potentially be affected. The last circuit modification that will be investigated is the use of transmission lines and if these lines will affect the DARPA TRUST output matching process. Table 1.1 shows the output goals for this research.

Table 1.1: Output Goals for Research

Experiment	Software Used	Goals
Testing Digital IP Cores	Cadence Encounter/Conformal	Test/verify/characterize output of TRUST program
Custom vs. Built-In circuits	Cadence Virtuoso	Determine if built-in circuits affect TRUST program
Extracting parasitic capacitance	Cadence Virtuoso	Determine if parasitic capacitance affects TRUST program
Implementing transmission lines	Cadence Virtuoso	Determine if transmission lines affect TRUST program

1.5 Assumptions and Scope

A successful test for this experiment will yield a match between the input and output product, (where the product in this case means the netlist). There are two outputs to the DARPA TRUST program, P_D and P_{FA}, which refer to the probability of detection and the probability of false alarm. For the DARPA TRUST program, the goal is to have P_D as close

to 1 as possible and P_{FA} to be as close to 0 as possible. When dealing with 3^{rd} party IP logic, it is assumed that the P_D metric is not considered. This is because it is assumed that malicious hardware has not been added to the actual logic code of the design. Therefore, the goal of the experiment is to have the metric, P_{FA}, be as close to 0 as possible. The verification process uses the Cadence Conformal software and the nodes in the netlists are compared. If all the nodes in the forward and reverse direction are mapped together than the circuit was not tampered with. However, if there is a difference between the two netlists than the DARPA TRUST program outputs the two metrics, P_D and P_{FA}, and gives the corresponding value to each metric.

A successful test for the circuit manipulation experiment will be analyzing the transistor-level netlist matching process. It may be determined that the DARPA TRUST program should start trusting circuits at a transistor level as well as a gate-level.

1.6 Materials and Equipment

This research will be conducted in the Air Force Research Laboratory (AFRL) Mixed Signals Design Center (MSDC) (Wright-Patterson Air Force Base (AFB), OH) and at AFIT in the VLSI lab. Most of this research will be conducted at Air Force Institute of Technology (AFIT) in the VLSI lab using the Cadence software and the MSDC will be used when the DARPA TRUST program is needed to verify the digital circuits. Testing for this research will require the use of outside, 3^{rd} party sources such as opencores.org for the logic of digital designs. The main software design tool will be Cadence and this software is available at the MSDC and the AFIT VLSI lab.

The equipment needed for this research is already provided at AFRL's MSDC and the AFIT VLSI lab. This includes all lab stations that are able to run both Linux and Windows machines, as well as sufficient hardware and software packages needed to complete this test. Since the Cadence design tools are the most important toolset for this experiment, the MSDC and the VLSI lab has ample software capabilities for the research be completed

over there. The Cadence Encounter software will help tackle the problem by synthesizing digital circuits and breaking them down to their netlists. Once the circuit and netlists have been created, the Cadence Conformal software will attempt to map the forward and reverse direction netlists to help verify circuit operation.

II. Background

As described in Chapter 1, there are unique challenges in verifying correct circuit operation in VLSI circuits. It is not viable for the DoD to participate in in-house fabrication since these manufacture centers are expensive and it is more viable to out-source this process. Due to manufacturing costs, most of these fabrication centers are located overseas where the profit margin is greater. This leads to a need to verify correct circuit operation since the DoD could potentially fabricate circuits under untrusted conditions. The DARPA TRUST program was initiated to combat this need.

2.1 DARPA TRUST

The DARPA Trusted for Integrated Circuits (TRUST) set forth to address the need to combat an unknown, highly technologically advanced enemy, interested in destroying or degrading military systems as well as collecting unauthorized intelligence by modifying or creating hardware between design and delivery. With the premise that enemy agents do exist in the world and would possess the motivation, opportunity, talent, manpower, and time to conduct operations against a nation's microelectronics resources: the threat is considered credible by the Defense Technical Information Center [8]. The DARPA TRUST program seeks to provide a quantifiable measurement that electronic components must meet given the provided specifications. Also, these specifications do not differ in such a way that would degrade the operation of the device, or provide unauthorized use. The DARPA TRUST program focused on the application of the tools and methodologies on custom designs in which the designer would have detailed knowledge in its construction. However, in digital circuit designs there is a reliance on 3^{rd} party IP that is provided via the foundry, an IP vendor, or freely distributed on the Internet (i.e. opencores.org) which is leveraged to create complex digital systems.

2.1.1 Definitions.

In the case of this research, there will be three different definitions of "TRUST" that will be applied. The first definition of "trust" when referring to electronic hardware comes from the National Semiconductor Corporation. The National Semiconductor Corporation defined "TRUST" as, "the ability of the Department of Defense to have confidence that a system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during its life cycle." [19]. The definition was originally intended to be applied to trusted software as it was composed by the DoD for the Trusted Software Initiative. Since there is a strong similarity between hardware and software trust problems, this definition of trust will be applied to this research [11].

The second definition of trust was provided by the Defense Trusted Integration Circuit Strategy (DTICS) memorandum, dated 10 October 2003, which led to the initiation of the Trusted Foundry (TF) program [22]. The DTICS defined "TRUST" as, "trust" is the ability to certify that designs sensitive to national security concerns are secure in the hands of a commercial manufacturer [22]. The definition is useful when dealing with hardware that is produced in high volume since commercial manufacturing is optimal but when dealing with custom hardware, commercial manufacturing is not used.

One last definition of "trust" comes from former Acting Under Secretary of Defense for Acquisition, Technology, and Logistics Michael Wynne, who stated in 2004 that "trust" is "the confidence in one's ability to secure national security systems by assessing the integrity of the people and processes used to design, generate, manufacture, and distribute national security critical components." [31].

2.1.2 Initial TRUST Program Case Study.

The challenge of verifying a digital circuit is immense. Just a few transistors in a field of millions transistors may be to blame, and these transistors are identical to its neighbors.

The interconnect and placement of hardware are imperative to locating malicious logic. The DARPA Microsystems Technology Office (MTO) TRUST Project Presentation uses the examples of a 64-bit adder containing two malicious insertions to identify the problem of verifying digital circuits. This example illustrates the difficulty in identifying and verifying digital circuits [8]. Figure 2.1 shows a layout of transistors identifying the malicious alterations.

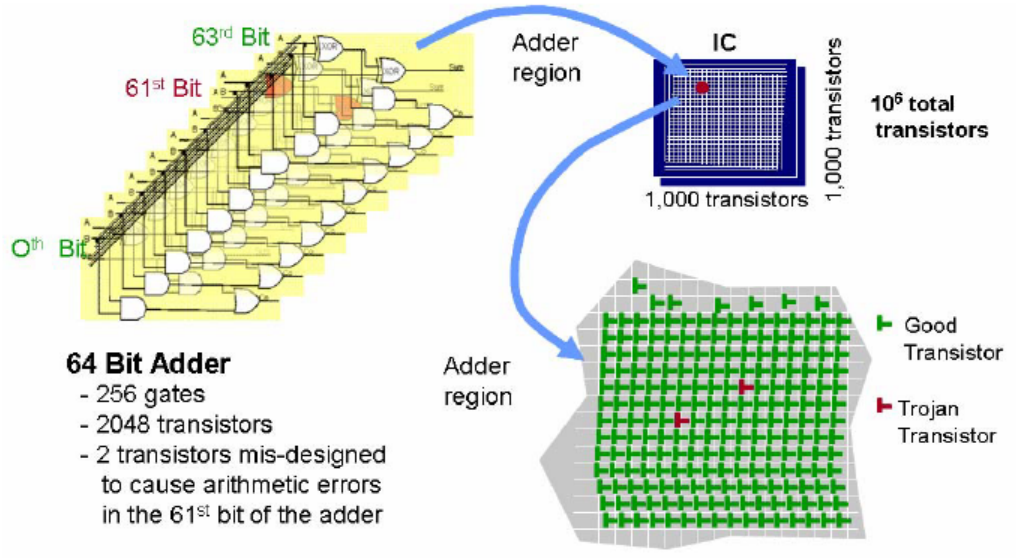

Figure 2.1: Metrics Challenge of 64-bit Adder [8]

The first transistor would cause an always-on state in an otherwise operational gate, while the second alteration is an event triggered condition for a specific type of adder input. The result of this experiment is an erroneous arithmetic output in the 61^{st} bit of one possible adder output at the cost of only two circuit modifications at the transistor level. The malicious insertion causes an erroneous output on the 61^{st} bit is because this is where the malicious transistors were inserted. Again, identifying these two malicious alterations provides a difficult problem. There are two quantifiable metrics that the DARPA TRUST tools output. First, the probablity of detection (P_D) refers to the probability of

correctly identifying malicious transistors. Meaning that a P_D value of 1 correctly identifies all malicious transistors inserted while a value of 0 did not identify any of the malicious alterations. Even malicious alterations to transistors can be operational, just not as they are designed to behave. P_D aims to identifying the number of incorrectly operating transistors in a circuit. The probability for a false alarm in malicious transistor detecting is defined as P_{FA}. P_{FA} refers to an operational transistor being identified as a malicious alteration by the DARPA TRUST tools. A P_{FA} value of 0 means that no operational transistors were identified as malicious, while a P_{FA} value close to 1 means many operational transistors were identified as malicious alterations.

Functional testing on the 64-bit adder can verify that the adder does not produce the correct output, but there is no way to locate the malicious insertion. This limitation is demonstrated in Fig. 2.2. Functional testing on that adder has $P_D = 1$, which is an excellent output for identifying unusable chips since the initial DARPA TRUST experiment identified that there are malicious transistors somewhere in the system. This P_D value of one means that all malicious transistors in the whole adder circuit were identified. However, P_{FA} for functional testing is unacceptably high. This means that the TF was unable to successfully locate the malicious transistor in the field of 10^6 transistors. The P_D and P_{FA} metrics are used to detect individual transistors and not used to detect inoperable devices (i.e. adders, accumulators, decoders, etc.). It is also important to note that functional testing will not necessarily identify all malicious logic in a circuit, if the insertion does not modify the current output of the device. This means that if a Hardware Trojan is put in place to trigger a bad output of a transistor based on lifetime, then the TF program will not initially identify the circuit as being bad. Functional testing has the added benefit of, at the system level, being a non-destructive, non-invasive test [18].

From an intelligence prospective, it is important to identify the physical location of the malicious transistors. Knowing how a Hardware Trojan was implemented allows an analyst

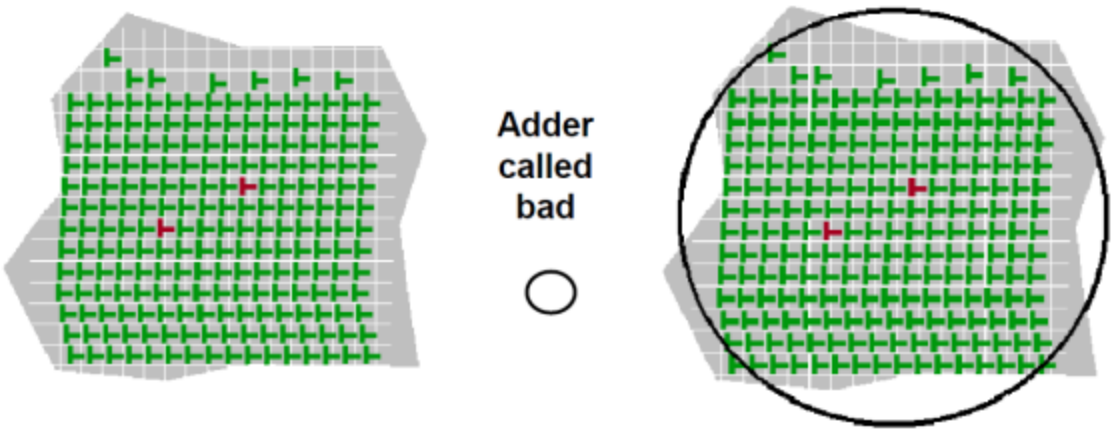

Figure 2.2: Functional test on example adder [8]

to predict future attack strategies as well as to identify the stage at which the trusted supply chain may have been compromised. By this means a proactive strategy can be implemented to combat hardware intrusions instead of merely waiting for an attack to occur. The DARPA TRUST program uses guided metrics for P_D, P_{FA}, problem size, and solution runtime. These results for the adder test are shown in table 2.1.

Table 2.1: DARPA TRUST Metric [8]

Metric	**Phase 1**	**Phase 2**	**Phase 3**
P_D	80.00%	90.00%	99.00%
P_{FA}	1.00E-03	1.00E-04	1.00E-06
Problem Size (Transistor Count)	1.00E+05	1.00E+06	5.00E+07
Algorithm Runtime (Hrs)	480	240	120

2.2 Factors Constraining DoD Demand of ICs

The DoD demand itself is very small in comparison to the private sector. For example, there are only 187 F-22 Raptors operational as of today, each of them including a multitude of custom components [13]. One hundred eighty seven is a small order size compared to the production volume of most desktop microprocessors. However, the DoD has sought to adhere to the DTICS memorandum definition of trust, which was originally intended to "trust" a commercial manufacturer. The DoD stated that defense technologies must improve at a similar rate to commercial devices, without regard for the decreased production volume, in order for the strategy to be effective [24]. In other words, defense technologies must be comparable in performance metrics to those devices used in the commercial sector.

Volume of hardware is just one of the factors constraining the hardware manufacturing of DoD based VLSI circuits. The Defense Microelectronics Activity (DMEA) quantified some of these factors, as shown in table 2.2, as a resource for the Professional Council of Federal Scientists and Engineers.

Table 2.2: Defense vs. Commercial Requirements [5]

	Commerical	Defense
System life span	less than 5 years	20 to 40 years
Quantities required	Very high volume (10^6 units)	Very low volume (10^2 to 10^3 units)
Fab production lifespan	2 years	Decades
Environmental	0 to 70 C	-55 to 125 C
Reliability/Quality	Lower, 10 years, non-hostile	High, hostile
Market share	greater than 90%	less than 0.1%

As one can see from Fig. **??**, the difference between commercial and defense systems varies greatly. While the system lifespan for a commercial system is less than five years, defense systems need to be operational for 20 to 40 years. For example, the Lockheed C-130 Hercules has been in service since 1956 and this aircraft is still flown today [4]. Even though the technology used in 1956 is different than that from today, the aircraft has a need to stay up to date with current technology and therefore custom hardware needs to be integrated into the aircraft longer than that of typical commercial products. Due to this fact, it is a greater importance for defense systems to be operating correctly and not tampered with. DARPA TRUST was established to verify correct circuit operation from untrusted sources and the C-130 is an aircraft where reliable circuits are a necessity. Another difference between commercial and defense systems is the reliability and quality of the systems. There is a critical need for defense systems to operate at a higher rate of reliability than standard commercial systems due to the risk of injury and/or death. Again, this means that verification of correct circuit operation is a necessity.

2.3 Quantifying Digital Diversity

This section outlines the need to quantify different digital components and process them through the DARPA TRUST suite. There is a huge need to test and verify the commercial IP cores that are available through open source. These cores are used by many commercial manufactures since they are open source. One example of a website online where these cores are available is through opensource.org. As of now, there has not been significant testing and verification of these IP cores through the TRUST suite. The purpose of this research is to perform a characteristic study of the different structures of these cores. Some examples of cores that would be tested are: cryptography cores, Fast-Fourier transform (FFT) cores, H.264 cores, etcetera.

The first step in the process is to identify which cores should be tested and why. The obvious starting point for this step is to identify which cores are used most in the

commercial manufacturing of circuits. For example, FFT cores are a key component in data manipulation between the time domain and frequency domain. After identifying common cores that are used in commercial manufacturing, the next step is to identify open cores that are of an interest to the DoD. The purpose of DARPA TRUST is to verify circuits used widely in the DoD. Once the cores have been identified the process becomes more complicated. There needs to be some sort of test plan in place that identifies different areas to stress with DARPA TRUST. These different areas can potentially include synthesizing the circuits under different constraints like speed, area, etc. This process will help identify how the different constraints affect how the DARPA TRUST suite verifies the circuit. The same circuit that is optimized for speed might have a different output from the DARPA TRUST suite than that same circuit optimized for area. The goal is to identify how the different constraints affect the output.

Another area that can affect how the DARPA TRUST output will be generated is the type of logic depth. The logical depth refers to the complexity of the logic in the system. For example, an finite impulse response (FIR) filter is composed of adders, multipliers, and registers. This type of logic is different from a cryptography core since the logic is different. The goal of this process is to see if different logic affects the output of the DARPA TRUST suite. The case might be that having a lot of multipliers in the design affects the DARPA TRUST suite negatively compared to a sequence of registers. The goal of this process is to identify if the different logic affects the verification process of the TRUST suite. Table 2.3 shows the estimate for standard cell usage, by type, of an Advanced Encryption Standard (AES) core.

Sequential logic refers to the logic whose output depends not only on the present value, but on the sequence of past events meaning this type of cell as some sort of memory. Inverter cell usage strictly refers to the number of inverter gates in the AES core. Finally, logic refers to all other logic in the AES core. This is comprised of combinational logic,

Table 2.3: AES Core Compiler Estimates

Cell Usage	Instances	% of Instances	Area	% of Area
Sequential	12736	4.4946517	298165.862	14.1
Inverter	29395	10.3737661	110619.264	5.2
Logic	241228	85.1315822	1701046.368	80.6
Total	**283359**	**100**	**2109831.494**	**100**

which unlike sequential logic, has no memory and the output only depends on the current value. As one can see from table 2.3, most of the area comes from the logic. One area of interest that can be investigated is how the output would change from a core that has most of the area coming from inverters compared with that from logic. Identifying this can help with the DARPA TRUST program so that one can know which commercial cores can be manufactured compared with those that should be manufactured from a trusted source.

2.3.1 Identification of IP Cores to Verify with DARPA TRUST.

One area of this research is to identify which cores to test with the DARPA TRUST program. The IP cores that are going to be processed through the DARPA TRUST program are open source, 3^{rd} party IP cores taken from the OpenCores website. These open source IP cores that are sometimes used in DoD systems. The DARPA TRUST program maintains library locations for DoD-owned IP, to include classified components [12]. In 2008, an estimated 9,356 counterfeit incidents were reported in the defense IC supply chain [10]. Figure 2.3 shows an outlook of counterfeiting incidents reported or suspected between 2005 and 2008. Figure 2.4 shows the distribution of these different types of counterfeit incidents.

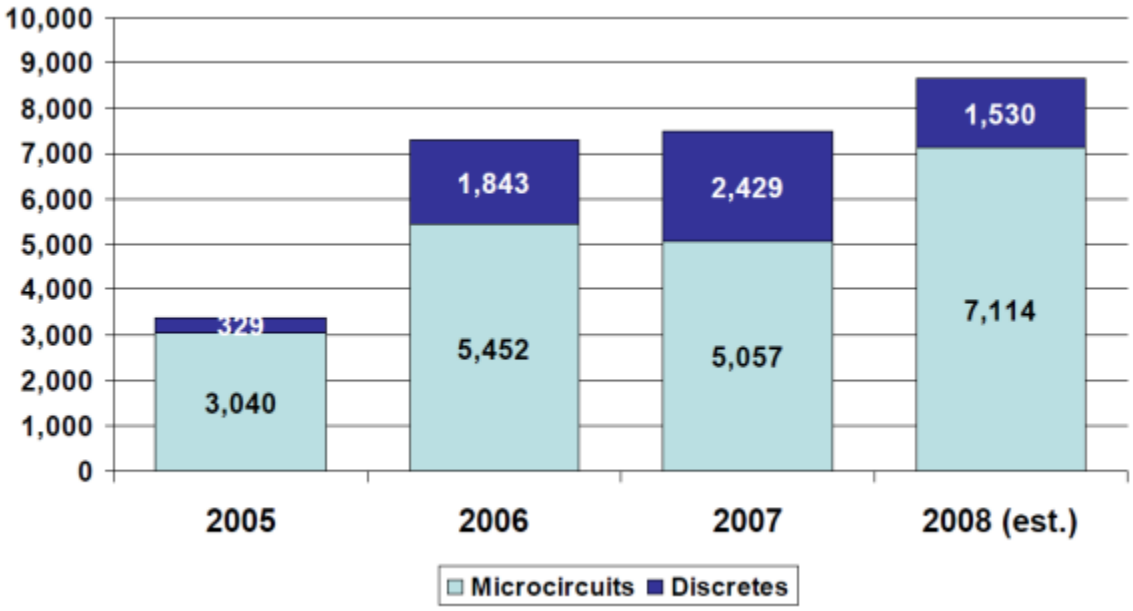

Figure 2.3: Total reported or suspected hardware counterfeits, 2005-2008 [21]

As one can see from Fig. 2.4, the most common type of counterfeited circuits are microprocessors. One hypothesis to explain this phenomenon is that microprocessors are prone to timing violations. This hypothesis derived from the fact that microprocessors can have multiple clocks running through them, and timing is a key role in the correct operation of microprocessors. Timing constraints are one way in which a circuit can be tampered. If a circuit fails to meet timing constraints, then the operation of the circuit will not behave as predicted. Another hypothesis to explain this phenomenon is that microprocessors are detected more often due to the relative high profit margin potential since microprocessors are used every where in society.

2.3.1.1 Timing Violations.

Almost all of digital integrated circuits work on the principle of synchrony. To summarize this statement, a common clock is used to synchronize all of its internal

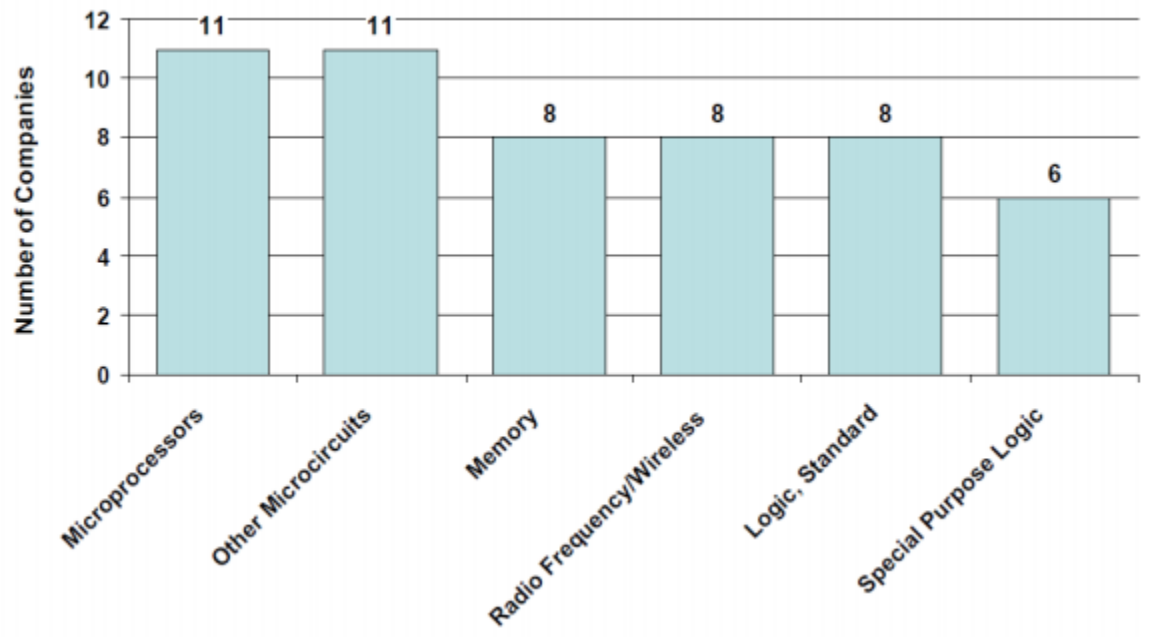

Figure 2.4: Companies reporting suspected or confirmed counterfeit microcircuits, by type [21]

operations. Figure 2.5 shows a basic representation of the internal architecture of a digital integrated circuit.

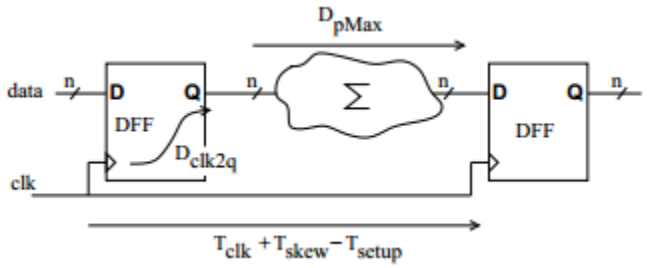

Figure 2.5: Internal architecture of digital ICs [32]

In order for timing constraints to be met, the clock period has to be longer in time compared with the maximum data propagation through the logic to ensure correct

operation. Two timing constraints that must always be met are the setup and hold time. Setup time is the amount of time data must be stabilized on the data line before the clock changes. Hold time is the amount of time that data must be held on the data line after the clock change. Figure 2.6 shows an outline of different timing constraints met and failed.

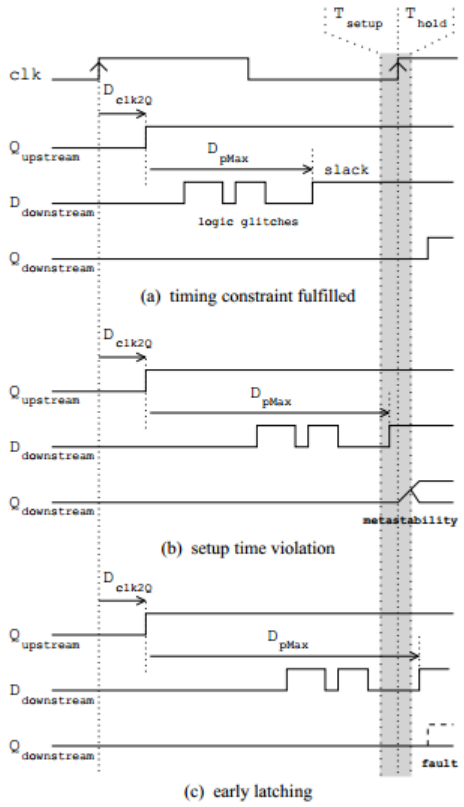

Figure 2.6: Timing constraint (a) fulfilled or violated: (b) setup violation, (c) early latching. [32]

The goal of summarizing timing constraints is to pick IP cores from the opencores website that are the most prone to failing timing violations. Running these circuits through the DARPA TRUST Foundry will help identify which open IP cores do not meet timing constraints and should be avoided.

2.4 Exploration of TRUST Suite

Even though the DARPA TRUST program was initiated in 2006, there still needs to be extensive research to discover the limitations and possible vulnerabilities of the program. The DARPA TRUST program operates at the gate-level of verifying circuits and the experiments conducted will be at the transistor-level. Bringing potential problems at the transistor-level to light could potentially change the way the DARPA TRUST program is conducted.

The DARPA TRUST tools are solving a very specific problem that is mapping transistors in a flattened netlist to those represented in the register-transfer level (RTL) reference. One area of exploration is how the parasitic capacitance in the metal layout affects the output of the DARPA TRUST program. Parasitic capacitance is an unavoidable capacitance that exists between the parts of a circuit due to the proximity to each other. This parasitic capacitance can cause transistors and metal wiring to depart from "ideal" conditions. In the case of this research, parasitic capacitance is introduced in the metal layout of a circuit. There is always a non-zero capacitance between any two conductors which can be significant at higher frequencies with closely space conductors, such as wires [16].Therefore, there will be parasitic capacitance in the metal wiring of the layout version of a circuit. For example, when a circuit designer creates a design, he or she creates a schematic version of the circuit and submits it to the manufacturer to fabricate the circuit. Most of the time, the circuit designer is not involved in this process, and therefore the parasitic capacitance of the metal is not taken into consideration during design. At this time, the DARPA TRUST program is not performing any experiments dealing with parasitic capacitance so exploring how parasitic capacitance affects the DARPA TRUST tools is a necessity.

The difference between the schematic version of an inverter and the metal layout of an inverter is shown in Fig. 2.7 and Fig. 2.8.

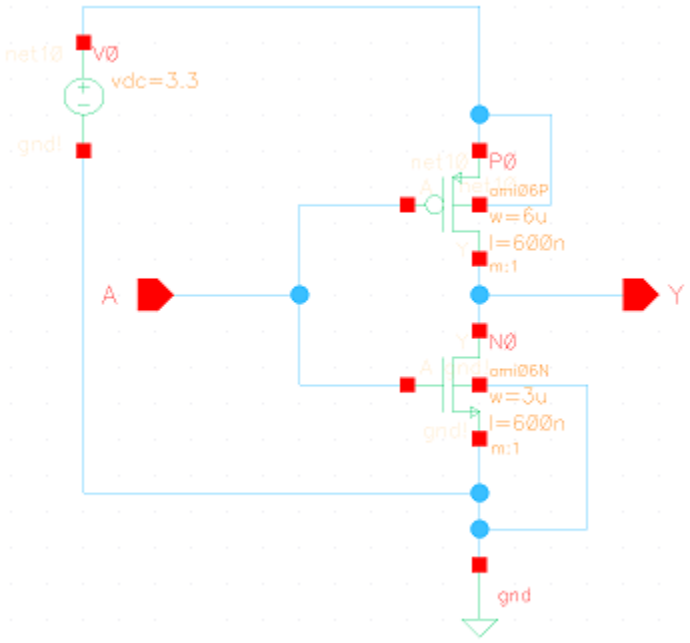

Figure 2.7: Schematic Inverter Design

As one can tell from Fig. 2.7 and Fig. 2.8, the difference between the layout and schematic version of the circuit is immense. When creating the metal layout version of a circuit, if the designer is not careful the parasitic capacitance can grow while the circuit is still operational. Another test in the exploration of the limitations of the DARPA TRUST program is the transmission lines between two digital devices. Figure 2.9 shows a schematic representation of this where the red circles represent the transmission lines. Transmission lines have the potential to cause errors in circuits if there is too much capacitance on the transmission lines.

Other areas of exploration include purposefully added short circuits in the metal layout to see how the DARPA TRUST program handles the flaw, increasing the power throughout the circuit to affect the netlist to see how the DARPA TRUST output changes, and using custom built circuits vs. built-in designs to see how the output changes.

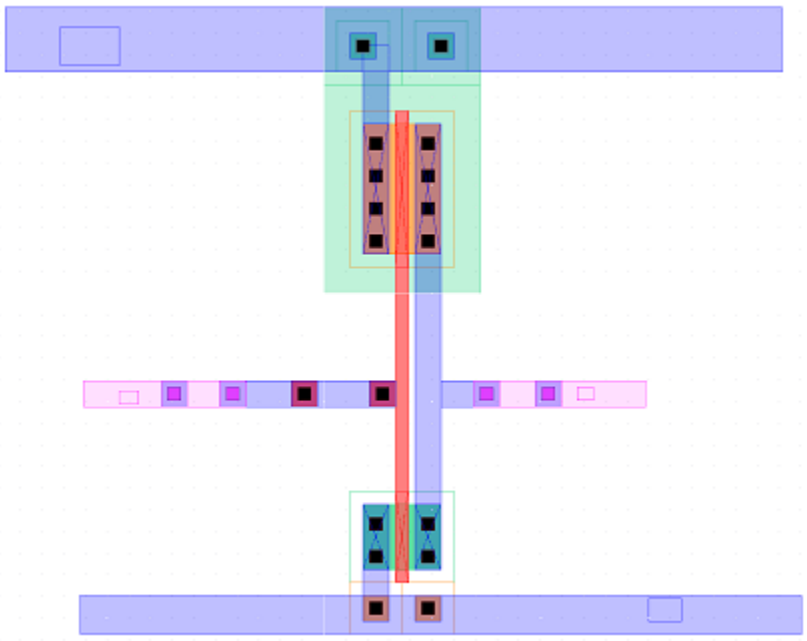

Figure 2.8: Metal Layout Inverter Design

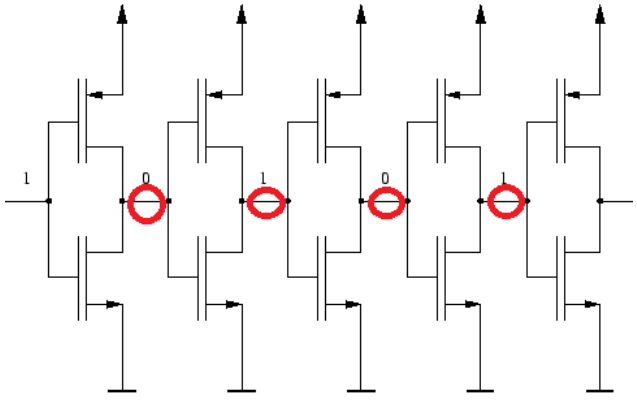

Figure 2.9: Series Inverter

2.5 Conclusion

The need to verify and "trust" VLSI circuits is an important challenge that the DoD faces. This research will provide more of an insight to the limitations of current programs and will help the warfighter in current and future operations.

III. Methodology

3.1 Introduction

As described earlier, this research focuses the limitations and possible errors in the DARPA TRUST program. Due to volume constraint problems, most of the digital components used in DoD systems are fabricated overseas where the cost is more economical. Potential threats associated with this phenomenon are what prompted the creation of the DARPA TRUST program. This phenomenon brought the DARPA TRUST program into implementation and is the reason why further research needs to be conducted on this program.

The first step in this research is the creation of a basic digital circuit from inception to fabrication and how the Cadence software tools will achieve this step. Digital circuit cores will then be examined and analyzed with the Cadence RTL compiler and Cadence Encounter tools. After that an equivalence check will be performed on the test circuits to see if the expected output matches the actual output. Finally, basic digital circuits will be manipulated to test the limitations and possible holes in the DARPA TRUST output.

3.2 Cadence Tools

The software used in this research is provided by Cadence. Cadence is an electronic design automation (EDA) software company that produces software for designing integrated circuits [15]. Cadence is used at a wide range of companies such as IBM and organizations such as the DoD. There are three components of the Cadence software that will be used in this research. The first product is Cadence Virtuoso. Cadence Virtuoso is a tool used for designing custom integrated circuits. This is the platform where schematics, custom layouts, extraction, and physical verification are performed. The second product is Cadence Encounter. The Encounter platform is a tool for the creation of digital integrated

circuits. This product is where floorplanning, synthesis, test, and place and route is performed. The last product is the RTL compiler. This is the platform where VHDL models are synthesized to be used for the Encounter program.

There are two libraries that will be used in this research. The most used library is the North Carolina State University (NCSU) process design kit (PDK). The attached technology to this library is 0.6 μm transistors. This transistor size is not up to date with the current technology size but for this research this technology will suffice. The second library used in this research is the University of Utah library. The attached technology for this library is also 0.6 μm transistors. This library will be used in the built-in circuit stage of this research.

There are different built-in functions in Cadence that will help achieve the end result of this research. In the Cadence Virtuoso product, custom schematics and layouts are able to be created. Once these are created Cadence has a layout vs. schematic (LVS) tool that can compare the netlists of the schematic and layout view. This LVS tool will be one of the main tools used in this research and will be discussed more later on in this paper.

3.3 Circuit Creation

3.3.1 Inception to Fabrication.

There is a specific flow of the TRUST tools that outlines the process from circuit specifications to fabrication [30]. Figure 3.1 shows the TRUST tool forward and reverse direction design flow.

This research will focus on Windows 1-3. Window 1 is where the 3^{rd} party IP cores will be implemented. These IP cores will be taken from the opencores.org website. VHDL code from this website will be synthesized in the RTL compiler Cadence product. Window 2 is where the synthesis and test insertion will take place. Synthesis of the VHDL code will be completed in the RTL compiler product. Test insertion is where a particular digital

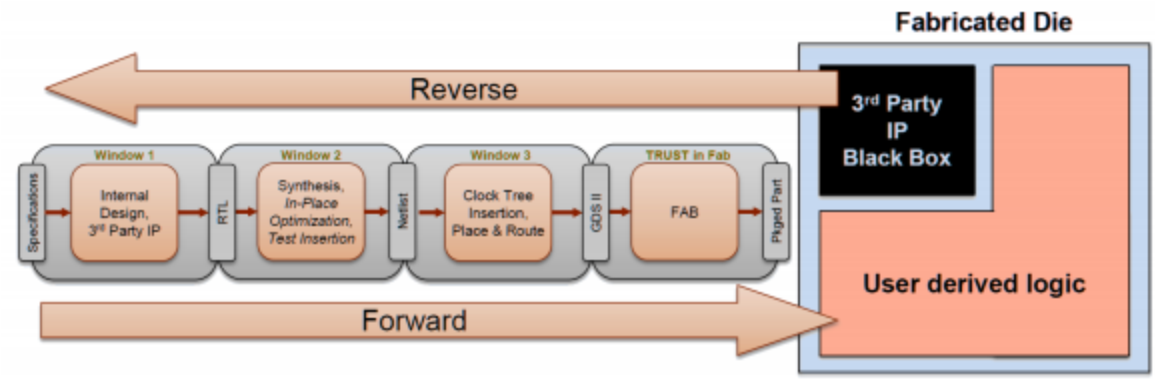

Figure 3.1: Forward and Reverse TRUST toolset design flow. [28]

component will be created. This will mostly be performed in the Cadence Virtuoso tool. Window 3 is where Clock Tree, Insertion, and Place & Route will take place. This window is accomplished in the Cadence Encounter software. Table 3.1 summarizes the correlation between the TRUST tool, application, and window according to Fig. 3.1.

Table 3.1: Table describing window correlation to Cadence product

Tool	Application	Window
Internal Design	RTL Compiler	Window 1
3^{rd} Party IP	RTL Compiler	Window 1
Synthesis	RTL Complier	Window 2
In-Place Optimization	RTL Compiler	Window 2
Test Insertion	Cadence Virtuoso	Window 2
Clock Tree Insertion	Cadence Encounter	Window 3
Place & Route	Cadence Encounter	Window 3

3.3.2 How Cadence tools will achieve thesis completion.

This section will outline how the Cadence tools will achieve thesis completion. The first step in circuit creation is to create a schematic based on certain specifications. These specifications can include the type of digital component needed to complete the design, the size of the transistors to be used in the component, and the pin names of the input and output pins.

The second step in circuit creation is to create a metal layout design based on the circuit schematic specification. NMOS and PMOS transistors are able to be inserted into the metal layout design to create a layout of a circuit. The Cadence Virtuoso layout editor is able to insert components needed for a metal layout design. These components include: nTaps, pTaps, metal vias, and metal wires.

3.4 Digital Circuit Cores

The Cadence software has the ability to synthesize and test digital circuits. For this research the digital circuit cores synthesized and tested will be 3^{rd} party IP cores taken from opencores.org.

3.4.1 RTL Complier.

The first step needed to synthesize and test digital circuit cores is to utilize the Cadence RTL compiler tool. The Cadence RTL compiler has the ability to synthesize VHDL or Verilog code to be used in the Cadence Encounter tool. The default language for the RTL compiler is verilog, however for this research VHDL code will be utilized since that is the preferred language of the author. Figure 3.2 shows a screenshot of VHDL code of a 4-bit accumulator.

Using the code from Fig. 3.2, the code is synthesized in the RTL compiler to produce a synthesized netlist to be used in the Cadence Encounter program. The synthesized netlist produced by the RTL compiler is shown in Fig. 3.3.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity accum is
  port(C, CLR : in  std_logic;
       D : in std_logic_vector(3 downto 0);
       Q : out std_logic_vector(3 downto 0));
end accum;
architecture archi of accum is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
          tmp <= "0000";
        elsif (C'event and C='1') then
          tmp <= tmp + D;
        end if;
    end process;
    Q <= tmp;
end archi;
```

Figure 3.2: Example Accumulator VHDL Code [1]

The process outlined in this section was not produced by the author but was rather introduced to show the methodology of the RTL compiler [3].

3.4.2 Cadence Encounter.

The Cadence Encounter program is used to develop complex digital circuits. The first step in developing complex digital circuits is to import a synthesized netlist from the RTL compiler. After that a library will be imported using the same technology as used in the RTL compiler as well as timing definitions according to the user. The next step is to develop the floorplan. This is where the global nets are connected and the power planning is decided. After that the Place & Route is performed on the finished floorplan, and the digital circuit is completed.

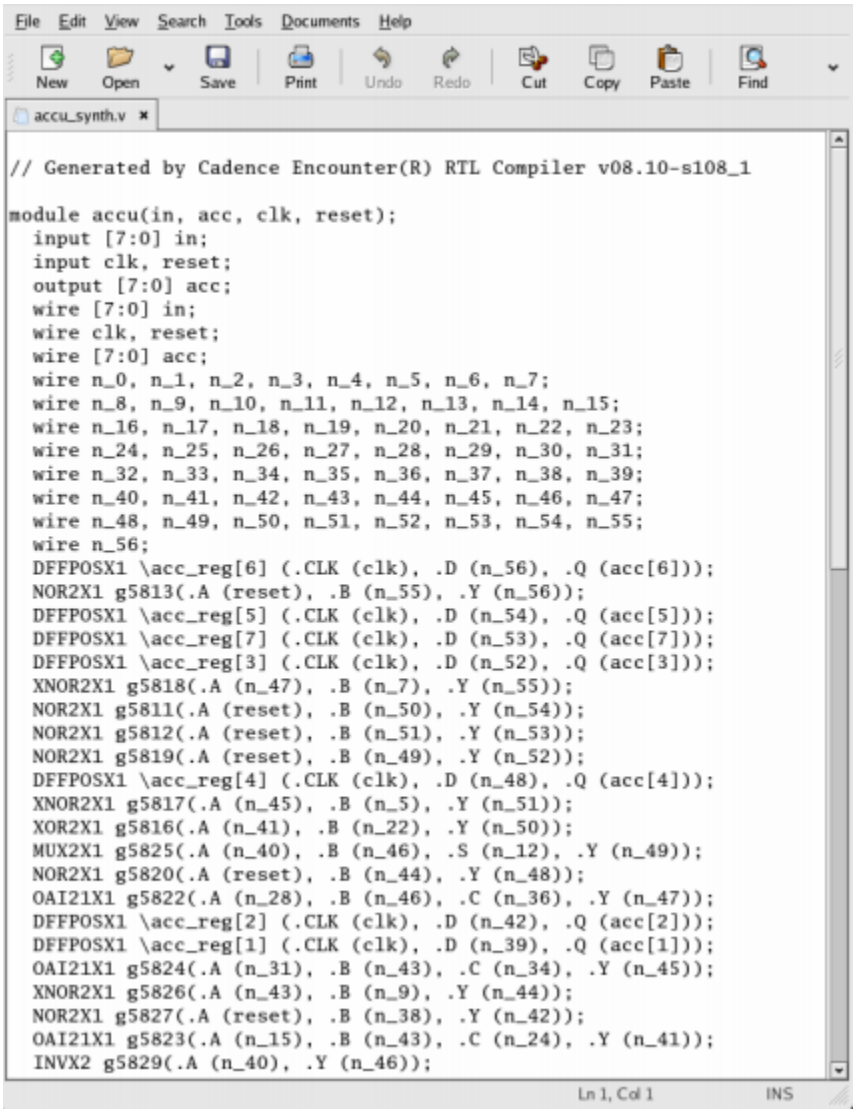

Figure 3.3: Example Synthesized Accumulator Netlist [2]

3.5 Matching Process

After the netlists are produced from each specific circuit the matching process will be applied. The goal of the matching process is to compare two netlists to see if they match. For this research, the netlist created from the schematic will be matched with the netlist created from the metal layout. The design tool used will be the LVS tool. The LVS tool imports netlists from both the schematic and metal layout of a circuit and then performs a matching process on the netlists to determine if they match. A matching result means that the two designs have the same internal architecture and therefore functionality.

Once the LVS tool is run the resulting output is an output log that describes the result of the program. This output log lists all of the nets from both of the netlists and lists their names. The output log says if the netlists match or do not match. If the netlists do not match then the output log counts how many nets do not match and tries to rewire the nets to produce a matching result. This is the end result of the research and determines if the circuit was "trusted" correctly.

3.6 Manipulating TRUST output

The DARPA TRUST program is a fairly new program. Due to this, there can be possible problems in the program that have not yet been explored. The first possible area that has not yet been explored is how custom vs. built-in circuits affect the netlist matching process. This will be accomplished by creating a custom schematic circuit and then perform the matching process with a built-in layout design. The built-in circuits used in this research will be taken from the University of Utah library. The circuits in this library range from basic digital components such as inverters to more complex digital components such as adders and multipliers. The custom schematic and layout circuits will be created using the methodology outlined in this chapter and the netlists will be matched with the matching process outlined in this chapter as well. This test will examine if different circuits with the same functionality will result in a successful match in the netlists. Another possible

area of unexplored territory is how parasitic capacitance affects the netlists and matching process. The parasitic capacitance of the metal wiring can either be ignored or included in the resulting netlist. This research will both extract and not extract parasitic capacitance to see if the netlist has any change at all. The output log produced will be examined to detect (if any) changes in the netlist. The last area of possible unexplored territory is how transmission lines affect the netlist. Transmission lines will be simulated by placing long metal wiring between two inverters. The metal wiring will be changed with each simulation to detect (if any) changes in the netlist. First, normal metal wiring will be placed between two inverters. After seeing the result of that simulation, a thick metal wiring will be placed between two inverters to see the result. Finally, a long metal wiring line will be placed between two spaced out inverters to detect changes.

3.7 Test Conditions

There will be five main tests that will be conducted in this research. These tests will be conducted at the transistor-level and will shed light on potential problems with the DARPA TRUST program only operating at the gate-level.

3.7.1 Initial Test.

The first test in this research will be called the "initial test." This test will create schematics and layouts for three different circuits: inverter, NAND2, and XOR2. The schematic and layout for each circuit will be matched using the LVS tool described in this chapter. The result of this initial test should produce a matching netlist result for all three circuits.

3.7.2 Custom vs. Built-In Circuits.

The second test in this research will compare the same three custom circuits from the initial test and compare them to built-in circuits derived from the University of Utah library. Like the initial test, the netlists will be compared with the Cadence LVS tool. The result

of this test will determine if the TRUST program is vulnerable to unmatched netlists when custom circuits are needed.

3.7.3 Parasitic Capacitance.

The next test in this research will be examining the effect of parasitic capacitance on the netlist of a circuit design. Parasitic capacitance can be extracted from the metal layout step in the process. Once the parasitic capacitance has been extracted the Cadence LVS tool will try and match the netlist with its resulting schematic. The result of this test will determine if the TRUST program is vulnerable to unmatched netlists when parasitic capacitance is extracted.

3.7.4 Transmission Lines.

The fourth test in this research will see if transmission lines has any affect on the netlists. Transmission lines will be simulated by placing a metal wire between two inverters of increasing size. The transmission line will be adjusted to differing lengths and widths to see if the netlist changes. Also, inverter sizing will be changed to see if that affects the netlist matching process. The result of this test will determine if the TRUST program is vulnerable to unmatched netlists when transmission lines are involved.

3.7.5 IP Cores.

The last test in this research will test and synthesize IP cores to test the limitations of the TRUST program. The IP cores will be synthesized in the Cadence RTL Compiler and then the resulting synthesized netlist will be imported into Cadence Encounter. In Encounter floorplanning, power distribution, and Place & Route will be conducted. Once all this happens, the Cadence matching process will be applied to see if the TRUST program is able to handle differing digital circuits. The result of this test will explore the limitations of the TRUST program.

IV. Results

This chapter introduces the results of the experiments carried out as outlined in Chapter 3. Each subsection of this chapter represents a different experiment that was conducted and includes an analysis of ways the DARPA TRUST program was explored and potentially areas where improvements can be made.

4.1 Initial Test

The initial test in this research was to perform a baseline experiment by creating a schematic and layout design of basic circuits. These circuits included an inverter, a NAND2 gate, and an XOR2 gate. Once these circuits were created, they were matched with the Cadence LVS tool.

4.1.1 Inverter.

The first test article was an inverter. Figure 4.1 shows the schematic version of an inverter created in Cadence while Fig. 4.2 shows the metal layout version of the same inverter. The width of the pMOS transistor was 6 μm, and the width of the nMOS transistor was 3 μm. Once the two versions of the inverter was created the Cadence LVS tool was run and created netlists for both versions as well as an output log showing the results. Figure 4.3 shows the netlists for both versions and Figure 4.4 shows the output log of the Cadence LVS tool. As expected, the netlists matched between the schematic and layout versions of the inverter.

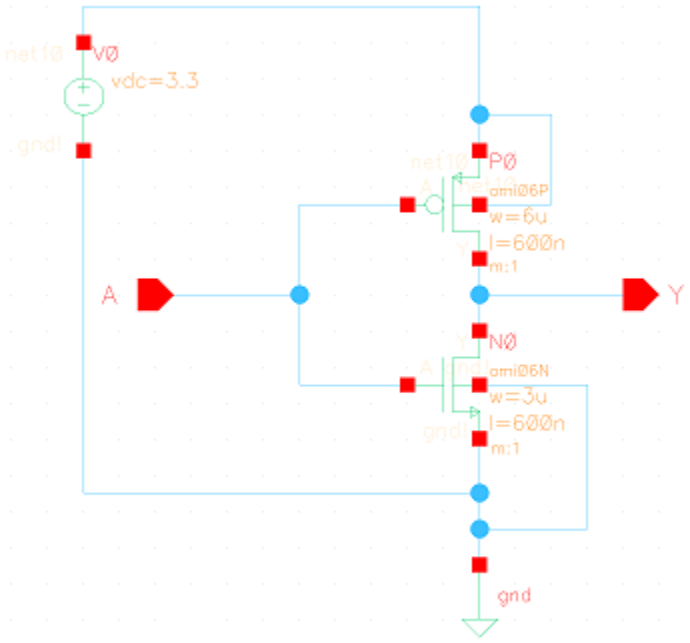

Figure 4.1: Inverter Schematic

4.1.2 NAND2.

The second test article was a NAND2 gate. Figure 4.5 shows the schematic version of a NAND2 gate created in Cadence while Fig. 4.6 shows the metal layout version of the same NAND2 gate. Once the two versions of the NAND2 was created, the Cadence LVS tool was run and created netlists for both versions as well as an output log showing the results. Figure 4.7 shows the netlists for both versions, and Figure 4.8 shows the output log of the Cadence LVS tool. As expected, the netlists matched between the schematic and layout versions of the NAND2. While the difference between the inverter and the NAND2 does not seem like a lot, the complexity increases in the number of nets in the circuit.

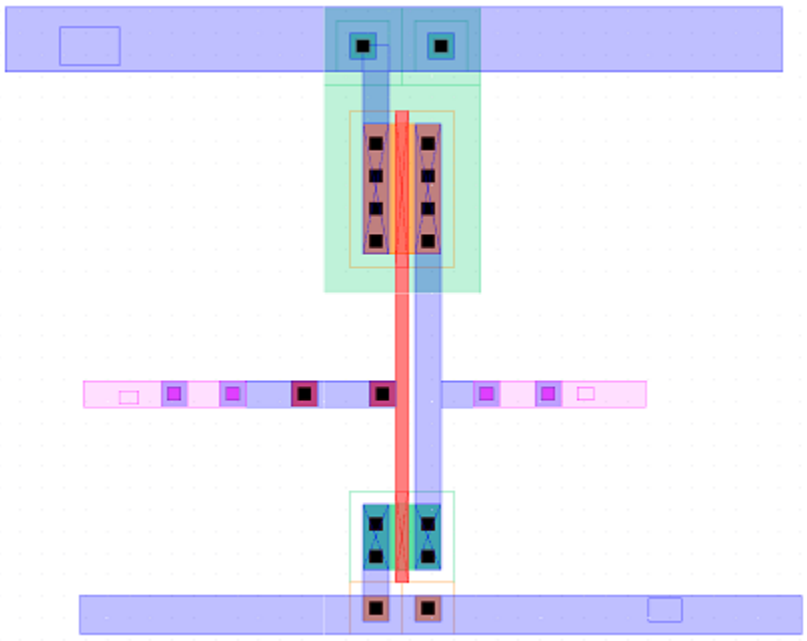

Figure 4.2: Inverter Metal Layout

Layout Netlist

```
t 1 Vin input
t 3 Vout output
t 0 gnd! inputOutput
t 2 vdd! inputOutput
n 0 /gnd!
n 1 /Vin
n 2 /vdd!
n 3 /Vout
; pmos4 Instance /+1 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 3 1 2 2 " m 1 l 600e-9 w 6e-6 "
; nmos4 Instance /+0 = auLvs device Q1
d nmos D G S B (p D S)
i 1 nmos 3 1 0 0 " m 1 l 600e-9 w 3e-6 "
```

Schematic Netlist

```
t 1 Vin input
t 3 Vout output
n 0 gnd!
n 1 /Vin
n 2 /net10
n 3 /Vout
; nmos4 Instance /N0 = auLvs device Q0
d nmos D G S B (p D S)
i 0 nmos 3 1 0 0 " m 1 l 600e-9 w 3e-6 "
; pmos4 Instance /P0 = auLvs device Q1
d pmos D G S B (p D S)
i 1 pmos 3 1 2 2 " m 1 l 600e-9 w 6e-6 "
t 0 gnd! Global
```

Figure 4.3: Inverter netlists

```
    Net-list summary for /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-
1.6.0.beta/LVS/layout/netlist
       count
       4             nets
       4             terminals
       1             pmos
       1             nmos

    Net-list summary for /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-
1.6.0.beta/LVS/schematic/netlist
       count
       4             nets
       3             terminals
       1             pmos
       1             nmos


    Terminal correspondence points
    N1         N1         Vin
    N3         N3         Vout
    N0         N0         gnd!

Devices in the rules but not in the netlist:
        cap nfet pfet nmos4 pmos4

The net-lists match.

                        layout  schematic
                          instances
       un-matched         0       0
       rewired                    0       0
       size errors        0       0
       pruned             0       0
       active             2       2
       total              2       2

                            nets
       un-matched         0       0
       merged             0       0
       pruned             0       0
       active             4       4
       total              4       4

                          terminals
       un-matched         0       0
       matched but
       different type             0       0
       total              4       3
```

Figure 4.4: Inverter Matching Process Output Log

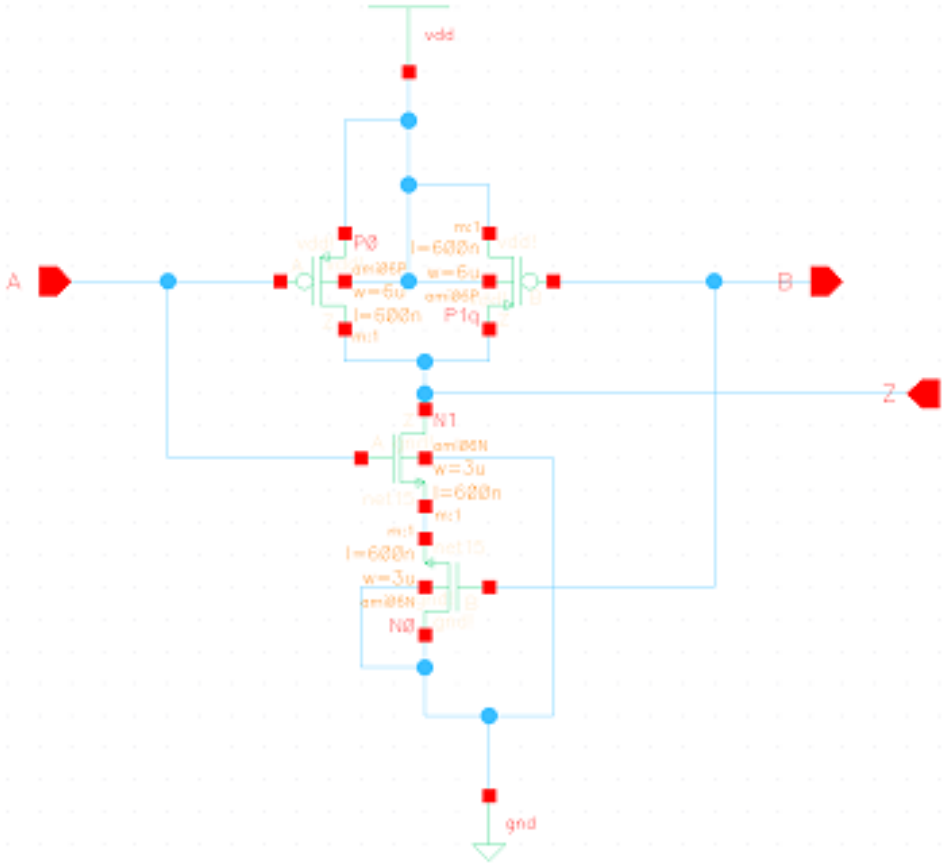

Figure 4.5: NAND2 Schematic

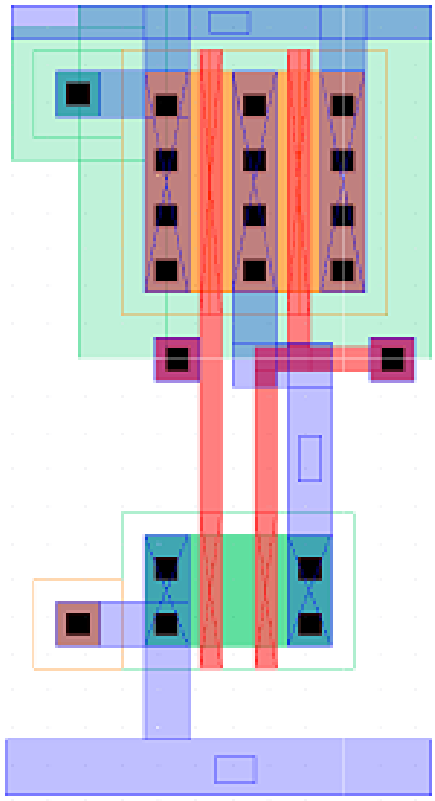

Figure 4.6: NAND2 Metal Layout

Layout Netlist

```
t 4 A inputOutput
t 3 B inputOutput
t 2 Z inputOutput
t 1 gnd! inputOutput
t 5 vdd! inputOutput
n 0 /6
n 1 /gnd!
n 2 /Z
n 3 /B
n 4 /A
n 5 /vdd!
; pmos4 Instance /+3 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 5 3 2 5 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+2 = auLvs device Q1
i 1 pmos 2 4 5 5 " m 1 l 600e-9 w 6e-6 "
; nmos4 Instance /+1 = auLvs device Q2
d nmos D G S B (p D S)
i 2 nmos 2 3 0 1 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+0 = auLvs device Q3
i 3 nmos 0 4 1 1 " m 1 l 600e-9 w 3e-6 "
```

Schematic Netlist

```
t 3 A input
t 2 B input
t 5 Z output
n 1 vdd!
n 0 gnd!
n 2 /B
n 3 /A
n 5 /Z
n 7 /net15
; nmos4 Instance /N1 = auLvs device Q0
d nmos D G S B (p D S)
i 0 nmos 5 3 7 0 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /N0 = auLvs device Q1
i 1 nmos 0 2 7 0 " m 1 l 600e-9 w 3e-6 "
; pmos4 Instance /P1q = auLvs device Q2
d pmos D G S B (p D S)
i 2 pmos 1 2 5 1 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /P0 = auLvs device Q3
i 3 pmos 5 3 1 1 " m 1 l 600e-9 w 6e-6 "
t 0 gnd! global
t 1 vdd! global
```

Figure 4.7: NAND2 netlists

4.1.3 XOR2.

The next test article was a XOR2 gate. Figure 4.9 shows the schematic version of a XOR2 gate created in Cadence, while Fig. 4.10 shows the metal layout version of the same XOR2 gate. Once the two versions of the XOR2 were created the Cadence LVS tool was run and created netlists for both versions as well as an output log showing the results. Figure 4.11 shows the netlists for both versions, and Figure 4.12 shows the output log of the Cadence LVS tool. As expected, the netlists matched between the schematic and layout versions of the XOR2. As one can see, creating a metal layout design of a XOR gate has a much higher complexity than the inverter gate. The netlists produced from the LVS tool is much more complicated than the inverter, and the runtime increased dramatically between the first and third experiment.

```
    Net-list summary for /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-
1.6.0.beta/LVS/layout/netlist
       count
       6             nets
       5             terminals
       2             pmos
       2             nmos

    Net-list summary for /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-
1.6.0.beta/LVS/schematic/netlist
       count
       6             nets
       5             terminals
       2             pmos
       2             nmos


    Terminal correspondence points
    N4         N3         A
    N3         N2         B
    N2         N5         Z
    N1         N0         gnd!
    N5         N1         vdd!

Devices in the rules but not in the netlist:
        cap nfet pfet nmos4 pmos4

The net-lists match.

                        layout  schematic
                         instances
       un-matched         0       0
       rewired                    0       0
       size errors        0       0
       pruned             0       0
       active             4       4
       total              4       4

                           nets
       un-matched         0       0
       merged             0       0
       pruned             0       0
       active             6       6
       total              6       6

                        terminals
       un-matched         0       0
       matched but
       different type             3       3
       total              5       5
```

Figure 4.8: NAND2 Matching Process Output Log

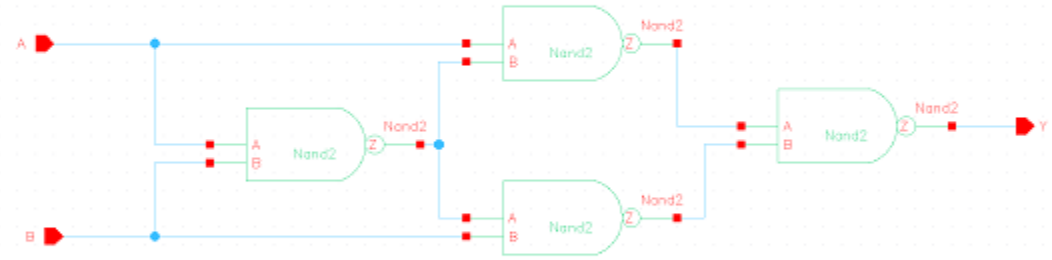

Figure 4.9: XOR2 Schematic

4.2 Custom vs. Built-In circuits

The next experiment conducted in this research was to compare custom designed circuits to built-in circuits from Cadence to see if the matching process still worked. This experiment would determine if a company could use their built-in circuit components in a circuit without the integrity of the circuit becoming compromised. There were initial problems that were run into due to the NCSU library used in this research. The pins were not labeled in the NCSU, library and this caused the netlists to not match. This brought up the issue of netlists matching because if the input, output, gnd, and vdd pins were not correctly labeled then the netlists did not match correctly. This could cause issue to the TRUST program because the verifier needs to make sure the pins are labeled correctly. Another issue during this experiment is that the transistor level view of the NCSU library

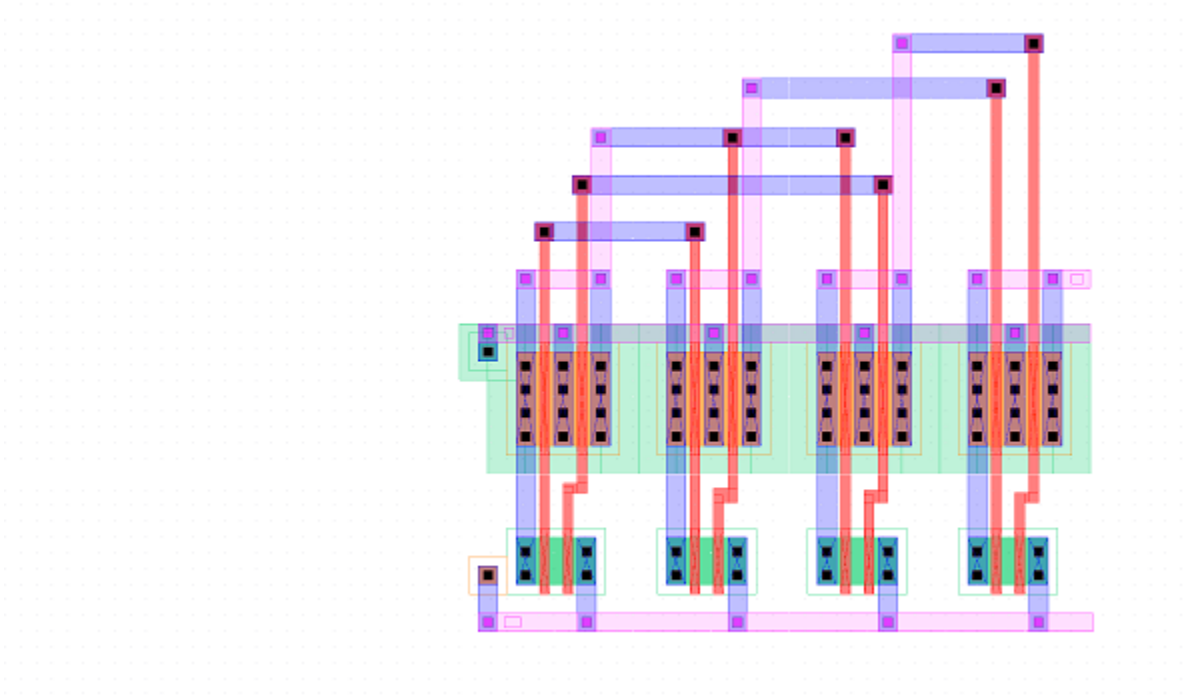

Figure 4.10: XOR2 Metal Layout

was not able to be examined, and therefore the netlist matching process was not able to work. Figure 4.13 shows the XOR2 gate that was used in this experiment.

The results of trying to conduct this experiment are shown in Fig. 4.14. The error in this experiment is that the schematic portion of the test was unable to flatten the netlist. The schematic XOR2 gate was taken from the built-in NCSU library. After further examination it was determined that a transistor level view of the XOR2 gate was not available and therefore the netlist could not be flattened.

Layout Netlist

```
t 10 A input
t 9 B input
t 8 Z output
t 7 gnd! inputOutput
t 11 vdd! inputOutput
n 0 /12
n 1 /11
n 2 /10
n 3 /9
n 4 /8
n 5 /7
n 6 /6
n 7 /gnd!
n 8 /Z
n 9 /B
n 10 /A
n 11 /vdd!
; pmos4 Instance /+15 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 8 4 11 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+14 = auLvs device Q1
i 1 pmos 11 5 8 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+13 = auLvs device Q2
i 2 pmos 4 9 11 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+12 = auLvs device Q3
i 3 pmos 11 6 4 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+11 = auLvs device Q4
i 4 pmos 5 6 11 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+10 = auLvs device Q5
i 5 pmos 11 10 5 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+9 = auLvs device Q6
i 6 pmos 6 9 11 11 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+8 = auLvs device Q7
i 7 pmos 11 10 6 11 " m 1 l 600e-9 w 6e-6 "
; nmos4 Instance /+7 = auLvs device Q8
d nmos D G S B (p D S)
i 8 nmos 7 4 0 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+6 = auLvs device Q9
i 9 nmos 0 5 8 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+5 = auLvs device Q10
i 10 nmos 7 9 1 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+4 = auLvs device Q11
i 11 nmos 1 6 4 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+3 = auLvs device Q12
i 12 nmos 7 6 2 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+2 = auLvs device Q13
i 13 nmos 2 10 5 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+1 = auLvs device Q14
i 14 nmos 7 9 3 7 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+0 = auLvs device Q15
i 15 nmos 3 10 6 7 " m 1 l 600e-9 w 3e-6 "
```

Schematic Netlist

```
t 2 A input
t 6 B input
t 7 Z output
n 1 vdd!
n 0 gnd!
n 2 /A
n 3 /net9
n 4 /net7
n 5 /net10
n 6 /B
n 7 /Z
n 13 /I3/net15
; nmos4 Instance /I3/N1 = auLvs device Q0
d nmos D G S B (p D S)
i 0 nmos 7 3 13 0 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /I3/N0 = auLvs device Q1
i 1 nmos 0 5 13 0 " m 1 l 600e-9 w 3e-6 "
; pmos4 Instance /I3/P1q = auLvs device Q2
d pmos D G S B (p D S)
i 2 pmos 1 5 7 1 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /I3/P0 = auLvs device Q3
i 3 pmos 7 3 1 1 " m 1 l 600e-9 w 6e-6 "
n 19 /I2/net15
; nmos4 Instance /I2/N1 = auLvs device Q4
i 4 nmos 5 4 19 0 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /I2/N0 = auLvs device Q5
i 5 nmos 0 6 19 0 " m 1 l 600e-9 w 3e-6 "
; pmos4 Instance /I2/P1q = auLvs device Q6
i 6 pmos 1 6 5 1 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /I2/P0 = auLvs device Q7
i 7 pmos 5 4 1 1 " m 1 l 600e-9 w 6e-6 "
n 25 /I1/net15
; nmos4 Instance /I1/N1 = auLvs device Q8
i 8 nmos 3 2 25 0 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /I1/N0 = auLvs device Q9
i 9 nmos 0 4 25 0 " m 1 l 600e-9 w 3e-6 "
; pmos4 Instance /I1/P1q = auLvs device Q10
i 10 pmos 1 4 3 1 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /I1/P0 = auLvs device Q11
i 11 pmos 3 2 1 1 " m 1 l 600e-9 w 6e-6 "
n 31 /I0/net15
; nmos4 Instance /I0/N1 = auLvs device Q12
i 12 nmos 4 2 31 0 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /I0/N0 = auLvs device Q13
i 13 nmos 0 6 31 0 " m 1 l 600e-9 w 3e-6 "
; pmos4 Instance /I0/P1q = auLvs device Q14
i 14 pmos 1 6 4 1 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /I0/P0 = auLvs device Q15
i 15 pmos 4 2 1 1 " m 1 l 600e-9 w 6e-6 "
t 0 gnd! global
t 1 vdd! global
```

Figure 4.11: XOR2 netlists

```
    Net-list summary for /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-
1.6.0.beta/LVS/layout/netlist
       count
       12            nets
       5             terminals
       8             pmos
       8             nmos

    Net-list summary for /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-
1.6.0.beta/LVS/schematic/netlist
       count
       12            nets
       5             terminals
       8             pmos
       8             nmos


    Terminal correspondence points
    N10         N2          A
    N9          N6          B
    N8          N7          Z
    N7          N0          gnd!
    N11         N1          vdd!

Devices in the rules but not in the netlist:
        cap nfet pfet nmos4 pmos4

The net-lists match.

                        layout  schematic
                          instances
       un-matched         0       0
       rewired                    0       0
       size errors        0       0
       pruned             0       0
       active             16      16
       total              16      16

                            nets
       un-matched         0       0
       merged             0       0
       pruned             0       0
       active             12      12
       total              12      12

                          terminals
       un-matched         0       0
       matched but
       different type             0       0
       total              5       5
```

Figure 4.12: XOR2 Matching Process Output Log

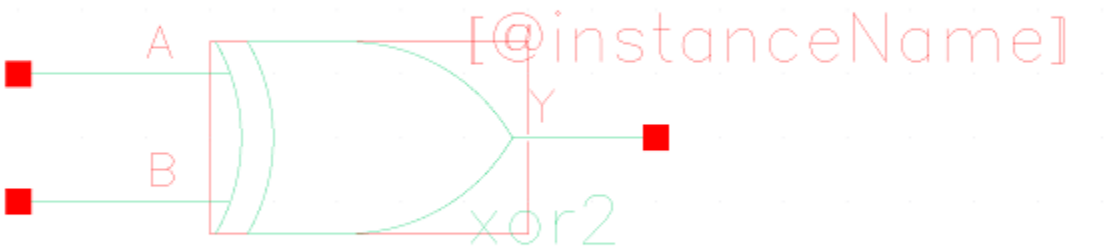

Figure 4.13: NCSU Library XOR2 Gate

The solution to this problem was to use the University of Utah library, and once this library was used the experiment was able to be conducted. Another issue that was discovered during this experiment is that even though two circuits have the same functionality, the netlist matching process has the potential to produce a non matching result. For example, there are many different ways to implement an XOR gate. If the internal structure of the custom XOR gate is different then that of the built-in XOR gate, the nets do not match up and therefore a non matching result is produced. Having a non matching result should not happen in this experiment due to the fact that the XOR2 gates have the same functionality. This is one area where having strictly gate-level testing in the DARPA TRUST program could be an issue. Figure 4.15 shows one implementation of an XOR2 gate.

```
Begin netlist:    Feb 12 12:10:57 2015
    view name list  = ("auLvs" "extracted" "schematic")
    stop name list  = ("auLvs")
    library name    = "EENG695"
    cell name   = "XOR"
    view name   = "extracted"
    globals lib = "basic"
Running Artist Flat Netlisting ...
*WARNING* Parameter 'lxUseCell' already exists.
End netlist:    Feb 12 12:10:57 2015

Moving original netlist to extNetlist
Removing parasitic components from netlist
    presistors removed:  0
    pcapacitors removed: 0
    pinductors removed:  0
    pdiodes removed:     0
    trans lines removed: 0
    12 nodes merged into 12 nodes


Begin netlist:    Feb 12 12:10:57 2015
    view name list  = ("auLvs" "schematic")
    stop name list  = ("auLvs")
    library name    = "NCSU_Digital_Parts"
    cell name   = "xor2"
    view name   = "symbol"
    globals lib = "basic"
Running Artist Flat Netlisting ...
global error:
The property 'connectivityLastUpdated' is not of type int on ( xor2 symbol ) in library NCSU_Digital_Parts.
    si: Netlist did not complete successfully.
End netlist:    Feb 12 12:10:57 2015
```

Figure 4.14: netlist Output Log from Initial Experiment

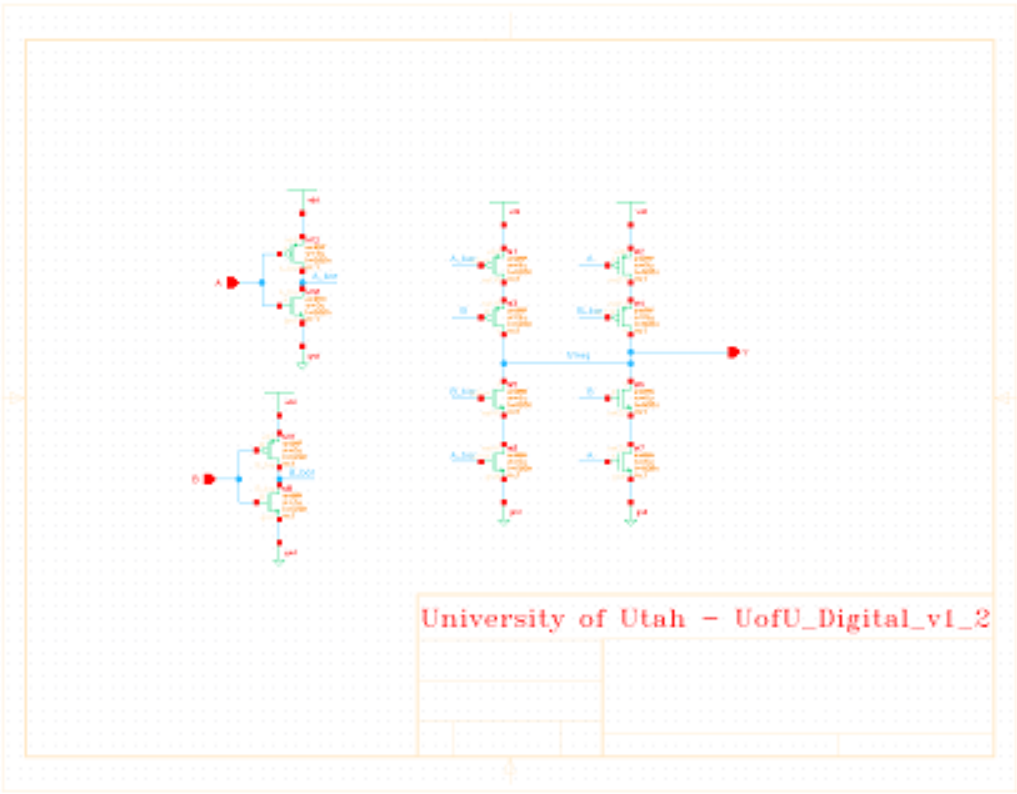

Figure 4.15: University of Utah Library XOR2 Gate

The implementation of this XOR2 gate is different than that of Fig. 4.9. Now a test was conducted to see of the matching process still worked even if the implementations of a XOR2 gate was different. Figure 4.16 shows the output log of this experiment.

```
The net-lists failed to match.

                    layout  schematic
                  instances
    un-matched         17  12
    rewired             0   0
    size errors         0   0
    pruned              0   0
    active             16  12
    total              16  12

                   nets
    un-matched          8   7
    merged              0   0
    pruned              0   0
    active             12  11
    total              12  11

                  terminals
    un-matched          1   1
    matched but
    different type          0   0
    total               5   5
```

Figure 4.16: netlist Output Log from XOR2 Custom Experiment

As one can tell from Fig. 4.16, the netlists failed to match. The first red-flag in this experiment is that the schematic and layout netlists had a different amount of nets. This means that the internal structure of the two versions of the XOR2 gate was different. Also, there was one terminal that was un-matched from both versions. These two issues caused the netlists to not match.

Since there was only one implementation of an XOR2 gate in the University of Utah library, the next test involves matching a custom vs. a built-in NAND2 gate. Figure 4.17 shows the Univeristy of Utah's implementation of a NAND2 gate.

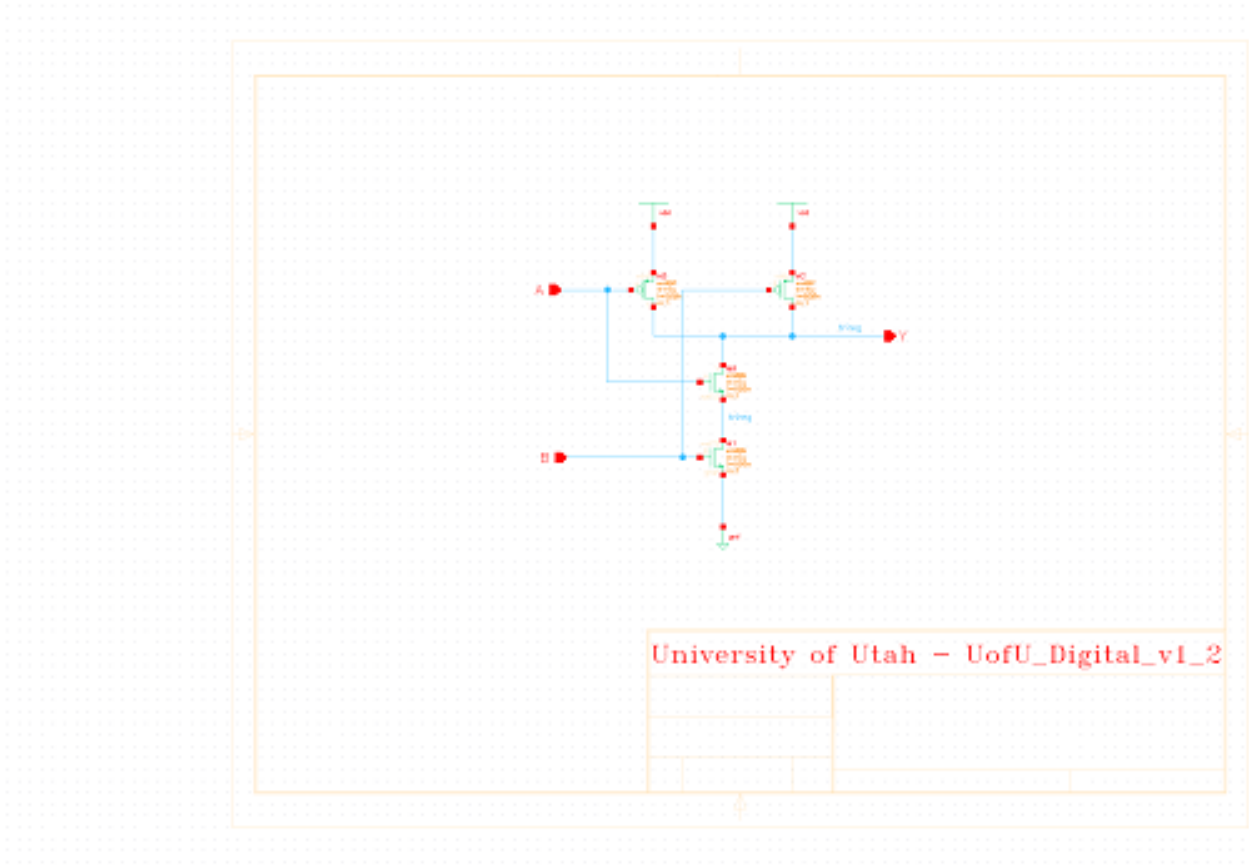

Figure 4.17: University of Utah Library NAND2 Gate

The implementation of the NAND2 gate shown in Fig. 4.17 is the same as Fig. 4.6. A test was conducted to see if the netlists matched from the custom NAND2 gate and the University of Utah NAND2 gate. Figure 4.18 shows the output log of this experiment.

As one can see from Fig. 4.18, the netlists from this experiment matched. This proved that as long as the internal architecture is the same, the netlists will match no matter if the gate is custom or taken from a library. Having the same internal architecture does not mean these two circuits are the exact same. The schematic version was taken from the University of Utah library. This circuit has different internal net names and the size of the transistors were not the same as the metal layout version. The metal layout version was designed by

```
The net-lists match.

                    layout  schematic
                  instances
    un-matched        0   0
    rewired           0   0
    size errors       0   0
    pruned            0   0
    active            4   4
    total             4   4

                     nets
    un-matched        0   0
    merged            0   0
    pruned            0   0
    active            6   6
    total             6   6

                  terminals
    un-matched        0   0
    matched but
    different type        3   3
    total             5   5
```

Figure 4.18: netlist Output Log from NAND2 Custom Experiment

the author and is based off the schematic of a NAND2 the author created. Comparing these two versions proved that circuits taken from a existing library can be used in the DARPA TRUST process as a custom metal layout was used in the comparison.

The final experiment conducted when comparing custom vs. built-in circuits was to change the pin names to see if the netlists still matched. The same NAND2 schematic taken from the University of Utah library shown in Fig. 4.17 was used, and that was compared to the metal layout NAND2 shown in Fig. 4.6. The only difference is that the output terminal in the metal layout version was changed from Y to Z. The netlists from the metal layout is shown in Fig. 4.19. The result of this experiment is shown in Fig. 4.20.

As one can see from Fig. 4.20, the netlists did not match. The internal structure of the schematic and metal layout version was the same, and therefore all of the nets matched. However, the terminals did not match and this caused the entire netlist to not match. This

```
t 4 A inputOutput
t 3 B inputOutput
t 2 Z inputOutput
t 1 gnd! inputOutput
t 5 vdd! inputOutput
n 0 /6
n 1 /gnd!
n 2 /Z
n 3 /B
n 4 /A
n 5 /vdd!
; pmos4 Instance /+3 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 5 3 2 5 " m 1 l 600e-9 w 6e-6 "
; pmos4 Instance /+2 = auLvs device Q1
i 1 pmos 2 4 5 5 " m 1 l 600e-9 w 6e-6 "
; nmos4 Instance /+1 = auLvs device Q2
d nmos D G S B (p D S)
i 2 nmos 2 3 0 1 " m 1 l 600e-9 w 3e-6 "
; nmos4 Instance /+0 = auLvs device Q3
i 3 nmos 0 4 1 1 " m 1 l 600e-9 w 3e-6 "
```

Figure 4.19: netlist from NAND2 Metal Layout Z Terminal

```
The net-lists failed to match.

                    layout  schematic
                  instances
    un-matched        0   0
    rewired           0   0
    size errors       0   0
    pruned            0   0
    active            4   4
    total             4   4

                     nets
    un-matched        0   0
    merged            0   0
    pruned            0   0
    active            6   6
    total             6   6

                  terminals
    un-matched        1   1
    matched but
    different type        2   2
    total             5   5
```

Figure 4.20: Output Log from NAND2 Metal Layout Z Terminal

test proved that the DARPA TRUST program would falsely flag two equivalent circuits as different due to a small alteration such as an output pin name.

The custom vs. built-in circuits experiment examined the limitations of trusting circuits. One must be careful when using built-in circuits as they can cause the netlists to not match if the designer does not pay attention to the internal structure and the pin names. The results from this experiment in the end were successful as built-in gates taken from a library are able to be used when trusting circuits.

4.3 Extracting parasitic capacitance

The next experiment conducted in this research was to examine how parasitic capacitance affects the netlist matching process. There was a possibility that parasitic capacitance in the metal layout version of a circuit could cause the netlist to differ so much from the schematic version of the same circuit that the netlists did not match. The first step in this experiment was to determine if there was a change in the netlists between the metal layout inverter circuit with and without extracting the parasitic capacitance. Figure 4.21 shows the netlists of an inverter with and without extracting parasitic capacitance. As one can see from Fig. 4.21, the netlist with the parasitic capacitance adds more complexity. However, this did not affect the LVS output matching process, and the netlists matched even with the parasitic capacitance added.

Inverter Netlist with parasitic capacitance

```
t 1 Vin input
t 3 Vout output
t 0 gnd! inputOutput
t 2 vdd! inputOutput
n 0 /gnd!
n 1 /Vin
n 2 /vdd!
n 3 /Vout
; pmos4 Instance /+1 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 3 1 2 2 " m 1 l 600e-9 w 6e-6 "
; pcapacitor Instance /+9 = auLvs device C1
d pcapacitor PLUS MINUS (p PLUS MINUS)
; pcapacitor Instance /+8 = auLvs device C2
; pcapacitor Instance /+7 = auLvs device C3
; pcapacitor Instance /+6 = auLvs device C4
; pcapacitor Instance /+5 = auLvs device C5
; pcapacitor Instance /+4 = auLvs device C6
; pcapacitor Instance /+3 = auLvs device C7
; pcapacitor Instance /+2 = auLvs device C8
; nmos4 Instance /+0 = auLvs device Q9
d nmos D G S B (p D S)
i 9 nmos 3 1 0 0 " m 1 l 600e-9 w 3e-6 "
```

Inverter Netlist without parasitic capacitance

```
t 1 Vin input
t 3 Vout output
t 0 gnd! inputOutput
t 2 vdd! inputOutput
n 0 /gnd!
n 1 /Vin
n 2 /vdd!
n 3 /Vout
; pmos4 Instance /+1 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 3 1 2 2 " m 1 l 600e-9 w 6e-6 "
; nmos4 Instance /+0 = auLvs device Q1
d nmos D G S B (p D S)
i 1 nmos 3 1 0 0 " m 1 l 600e-9 w 3e-6 "
```

Figure 4.21: Inverter netlists with and without parasitic capacitance

The next step when experimenting with parasitic capacitance was to test the extracted parasitic capacitance with a more complex circuit. The test article in this case was the XOR2 gate. The same process used with the inverter test was applied to the XOR2 gate, and the resulting netlists are shown in Fig. 4.22. Comparing Fig. 4.22 from Fig. 4.11, it is shown that parasitic capacitance adds lines to the netlist. Like the inverter, extracting the parasitic capacitance had an effect on the netlist. The parasitic capacitance netlist for this experiment added a lot more complexity for the netlist but as with the case of the inverter the LVS matching process was not affected as the layout and schematic versions of the XOR2 gate matched.

XOR2 Netlist with parasitic capacitance

```
t 10 A input
t 9 B input
t 8 Z output
t 7 gnd! inputOutput
t 11 vdd! inputOutput
n 0 /12
n 1 /11
n 2 /10
n 3 /9
n 4 /8
n 5 /7
n 6 /6
n 7 /gnd!
n 8 /Z
n 9 /B
n 10 /A
n 11 /vdd!
; pmos4 Instance /+15 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 8 4 11 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+14 = auLvs device Q1
i 1 pmos 11 5 8 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+13 = auLvs device Q2
i 2 pmos 4 9 11 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+12 = auLvs device Q3
i 3 pmos 11 6 4 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+11 = auLvs device Q4
i 4 pmos 5 6 11 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+10 = auLvs device Q5
i 5 pmos 11 10 5 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+9 = auLvs device Q6
i 6 pmos 6 9 11 11 " m 1 1 600e-9 w 6e-6 "
; pmos4 Instance /+8 = auLvs device Q7
i 7 pmos 11 10 6 11 " m 1 1 600e-9 w 6e-6 "
; pcapacitor Instance /+57 = auLvs device C8
d pcapacitor PLUS MINUS (p PLUS MINUS)
; pcapacitor Instance /+56 = auLvs device C9
; pcapacitor Instance /+55 = auLvs device C10
; pcapacitor Instance /+54 = auLvs device C11
; pcapacitor Instance /+53 = auLvs device C12
; pcapacitor Instance /+52 = auLvs device C13
; pcapacitor Instance /+51 = auLvs device C14
; pcapacitor Instance /+50 = auLvs device C15
; pcapacitor Instance /+49 = auLvs device C16
; pcapacitor Instance /+48 = auLvs device C17
; pcapacitor Instance /+47 = auLvs device C18
; pcapacitor Instance /+46 = auLvs device C19
; pcapacitor Instance /+45 = auLvs device C20
; pcapacitor Instance /+44 = auLvs device C21
; pcapacitor Instance /+43 = auLvs device C22
; pcapacitor Instance /+42 = auLvs device C23
; pcapacitor Instance /+41 = auLvs device C24
; pcapacitor Instance /+40 = auLvs device C25
; pcapacitor Instance /+39 = auLvs device C26
; pcapacitor Instance /+38 = auLvs device C27
; pcapacitor Instance /+37 = auLvs device C28
; pcapacitor Instance /+36 = auLvs device C29
; pcapacitor Instance /+35 = auLvs device C30
; pcapacitor Instance /+34 = auLvs device C31
; pcapacitor Instance /+33 = auLvs device C32
; pcapacitor Instance /+32 = auLvs device C33
; pcapacitor Instance /+31 = auLvs device C34
; pcapacitor Instance /+30 = auLvs device C35
; pcapacitor Instance /+29 = auLvs device C36
; pcapacitor Instance /+28 = auLvs device C37
; pcapacitor Instance /+27 = auLvs device C38
; pcapacitor Instance /+26 = auLvs device C39
; pcapacitor Instance /+25 = auLvs device C40
; pcapacitor Instance /+24 = auLvs device C41
; pcapacitor Instance /+23 = auLvs device C42
; pcapacitor Instance /+22 = auLvs device C43
; pcapacitor Instance /+21 = auLvs device C44
; pcapacitor Instance /+20 = auLvs device C45
; pcapacitor Instance /+19 = auLvs device C46
; pcapacitor Instance /+18 = auLvs device C47
; pcapacitor Instance /+17 = auLvs device C48
; pcapacitor Instance /+16 = auLvs device C49
; nmos4 Instance /+7 = auLvs device Q50
d nmos D G S B (p D S)
i 50 nmos 7 4 0 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+6 = auLvs device Q51
i 51 nmos 0 5 8 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+5 = auLvs device Q52
i 52 nmos 7 9 1 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+4 = auLvs device Q53
i 53 nmos 1 6 4 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+3 = auLvs device Q54
i 54 nmos 7 6 2 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+2 = auLvs device Q55
i 55 nmos 2 10 5 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+1 = auLvs device Q56
i 56 nmos 7 9 3 7 " m 1 1 600e-9 w 3e-6 "
; nmos4 Instance /+0 = auLvs device Q57
i 57 nmos 3 10 6 7 " m 1 1 600e-9 w 3e-6 "
```

Figure 4.22: XOR2 netlists with parasitic capacitance

Even though extracting parasitic capacitance did not have an affect on the LVS matching process, this does not mean that this should not be a concern for the DARPA TRUST program. When looking at the log for the LVS run, it was discovered that the parasitic capacitance lines were removed from the netlists. This means that the matching process did not even take this into consideration even though this could cause issues with the actual design of a real-life circuit. The issues that come with parasitic capacitance will not necessarily affect the netlist matching process. Running this experiment should be a matter of testing the actual functional design and not trying to verify circuit operation with netlists. This is another area that the DARPA TRUST program overlooks because the program is strictly looking at netlist matching. Future work could potentially take this feature out of the rules file to keep the parasitic capacitance in the matching process.

4.4 Transmission Lines

Examining how transmission lines could potentially affect the DARPA TRUST program was the next experiment conducted in this research. In this research, the transmission line was metal wiring between two inverters. The inverters varied in size to determine if this had any affect on the netlists. Also, the length and the width of the metal wire between the two inverters was changed to see if this had any affect on the output. When dealing with parasitic capacitance, the longer and wider the wider the metal wire, the more parasitic capacitance produced. There were four different test cases that were used in this experiment. Figures 4.23, 4.24, 4.25, and 4.26 shows the four different configurations. The four different configurations were a normal transmission line, a wide transmission line, a long transmission line, and a long and wide transmission line. The goal of this experiment was to determine if the netlists vary at all between the four different configurations. Figure 4.27 shows the four netlists corresponding to the four different configurations.

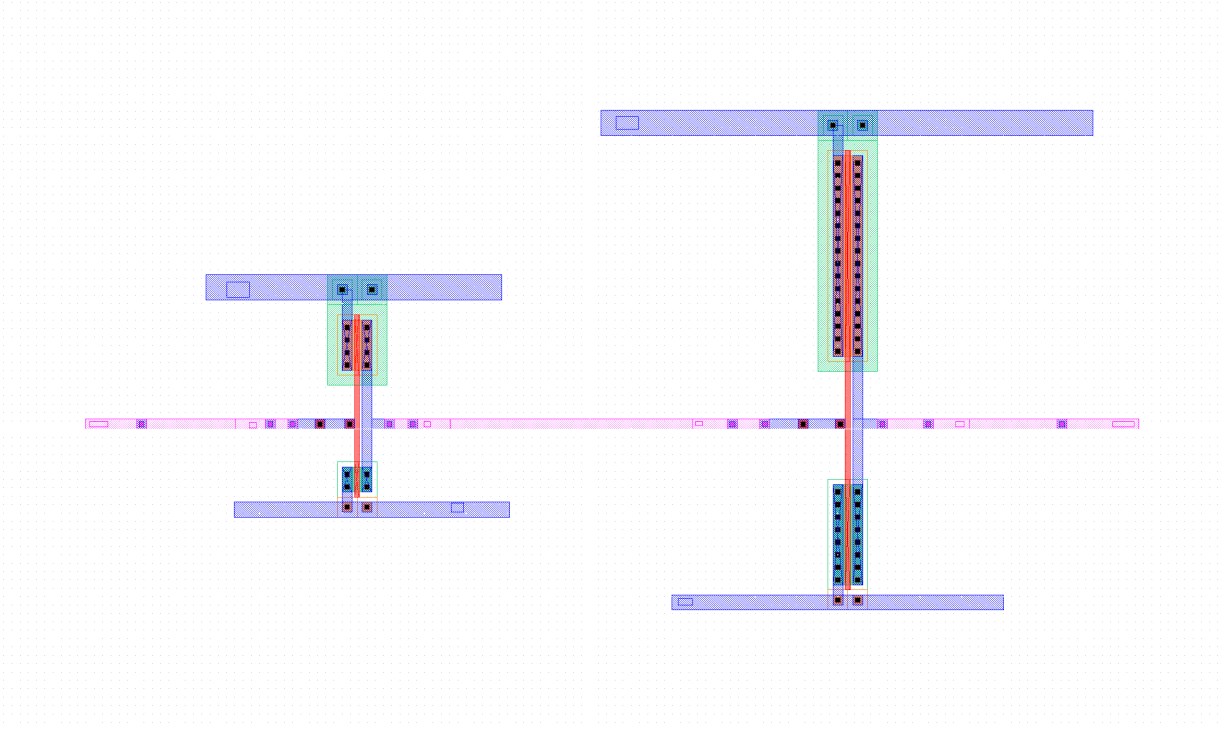

Figure 4.23: Normal Transmission Line

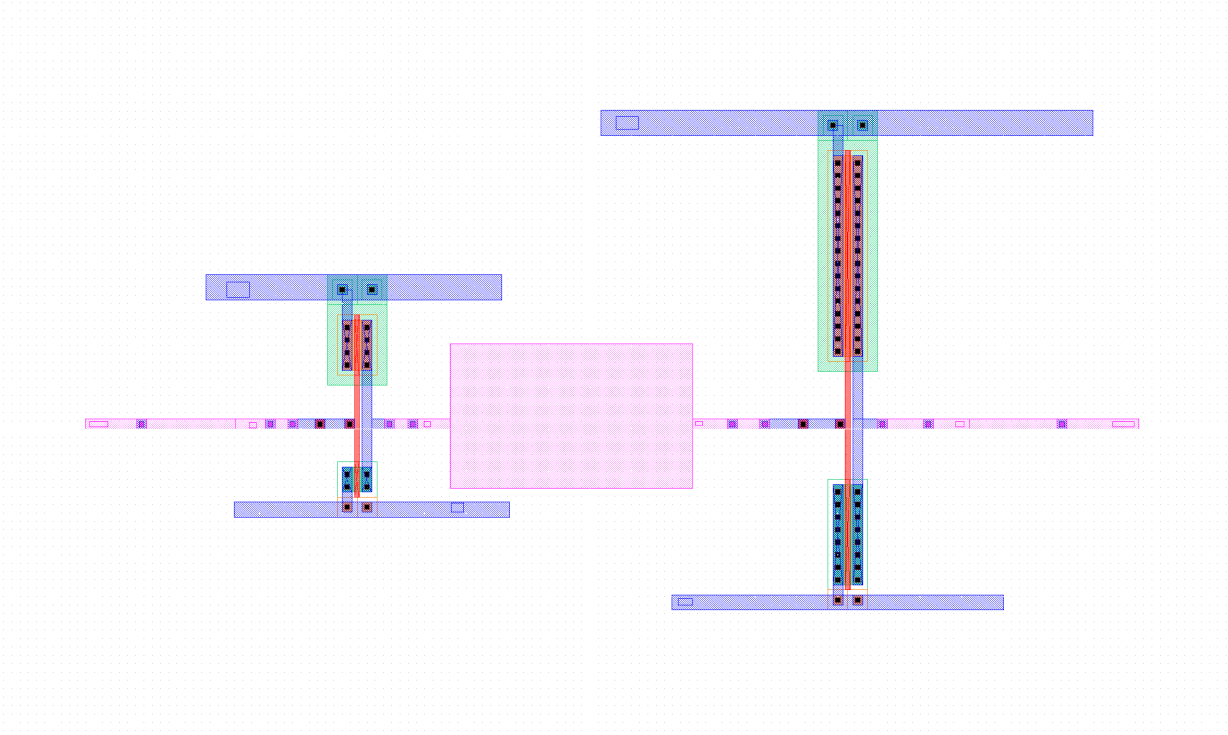

Figure 4.24: Wide Transmission Line

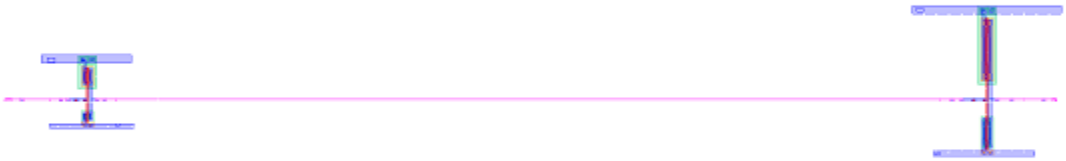

Figure 4.25: Long Transmission Line

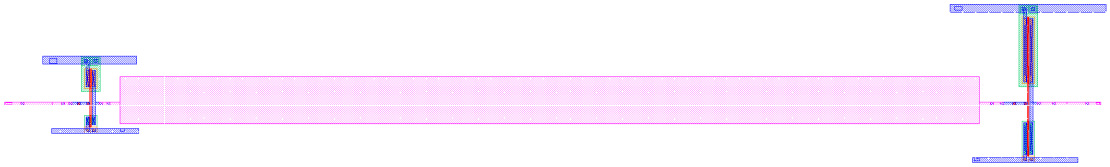

Figure 4.26: Long and Wide Transmission Line

Long Transmission Net-List

```
t 5 In input
t 4 Out output
n 0 /6
n 1 /5
n 2 /4
n 3 /3
n 4 /Out
n 5 /In
; pmos4 Instance /+3 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 4 0 2 2 " m 1 1 600e-9 w 24e-6 "
; pcapacitor Instance /+17 = auLvs device C1
d pcapacitor PLUS MINUS (p PLUS MINUS)
; pcapacitor Instance /+16 = auLvs device C2
; pcapacitor Instance /+15 = auLvs device C3
; pcapacitor Instance /+14 = auLvs device C4
; pcapacitor Instance /+13 = auLvs device C5
; pcapacitor Instance /+12 = auLvs device C6
; pcapacitor Instance /+11 = auLvs device C7
; pcapacitor Instance /+10 = auLvs device C8
; pcapacitor Instance /+9 = auLvs device C9
; pcapacitor Instance /+8 = auLvs device C10
; pcapacitor Instance /+7 = auLvs device C11
; pcapacitor Instance /+6 = auLvs device C12
; pcapacitor Instance /+5 = auLvs device C13
; pcapacitor Instance /+4 = auLvs device C14
; nmos4 Instance /+1 = auLvs device Q15
d nmos D G S B (p D S)
i 15 nmos 4 0 1 1 " m 1 1 600e-9 w 12e-6 "
; nmos4 Instance /+0 = auLvs device Q16
i 16 nmos 0 5 1 1 " m 1 1 600e-9 w 3e-6 "
; pmos4 Instance /+2 = auLvs device Q17
i 17 pmos 0 5 3 3 " m 1 1 600e-9 w 6e-6 "
```

Wide Transmission Net-list

```
t 5 In input
t 4 Out output
n 0 /6
n 1 /5
n 2 /4
n 3 /3
n 4 /Out
n 5 /In
; pmos4 Instance /+3 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 4 0 2 2 " m 1 1 600e-9 w 24e-6 "
; pcapacitor Instance /+17 = auLvs device C1
d pcapacitor PLUS MINUS (p PLUS MINUS)
; pcapacitor Instance /+16 = auLvs device C2
; pcapacitor Instance /+15 = auLvs device C3
; pcapacitor Instance /+14 = auLvs device C4
; pcapacitor Instance /+13 = auLvs device C5
; pcapacitor Instance /+12 = auLvs device C6
; pcapacitor Instance /+11 = auLvs device C7
; pcapacitor Instance /+10 = auLvs device C8
; pcapacitor Instance /+9 = auLvs device C9
; pcapacitor Instance /+8 = auLvs device C10
; pcapacitor Instance /+7 = auLvs device C11
; pcapacitor Instance /+6 = auLvs device C12
; pcapacitor Instance /+5 = auLvs device C13
; pcapacitor Instance /+4 = auLvs device C14
; nmos4 Instance /+1 = auLvs device Q15
d nmos D G S B (p D S)
i 15 nmos 4 0 1 1 " m 1 1 600e-9 w 12e-6 "
; nmos4 Instance /+0 = auLvs device Q16
i 16 nmos 0 5 1 1 " m 1 1 600e-9 w 3e-6 "
; pmos4 Instance /+2 = auLvs device Q17
i 17 pmos 0 5 3 3 " m 1 1 600e-9 w 6e-6 "
```

Long and Wide Transmission Net-list

```
t 5 In input
t 4 Out output
n 0 /6
n 1 /5
n 2 /4
n 3 /3
n 4 /Out
n 5 /In
; pmos4 Instance /+3 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 4 0 2 2 " m 1 1 600e-9 w 24e-6 "
; pcapacitor Instance /+17 = auLvs device C1
d pcapacitor PLUS MINUS (p PLUS MINUS)
; pcapacitor Instance /+16 = auLvs device C2
; pcapacitor Instance /+15 = auLvs device C3
; pcapacitor Instance /+14 = auLvs device C4
; pcapacitor Instance /+13 = auLvs device C5
; pcapacitor Instance /+12 = auLvs device C6
; pcapacitor Instance /+11 = auLvs device C7
; pcapacitor Instance /+10 = auLvs device C8
; pcapacitor Instance /+9 = auLvs device C9
; pcapacitor Instance /+8 = auLvs device C10
; pcapacitor Instance /+7 = auLvs device C11
; pcapacitor Instance /+6 = auLvs device C12
; pcapacitor Instance /+5 = auLvs device C13
; pcapacitor Instance /+4 = auLvs device C14
; nmos4 Instance /+1 = auLvs device Q15
d nmos D G S B (p D S)
i 15 nmos 4 0 1 1 " m 1 1 600e-9 w 12e-6 "
; nmos4 Instance /+0 = auLvs device Q16
i 16 nmos 0 5 1 1 " m 1 1 600e-9 w 3e-6 "
; pmos4 Instance /+2 = auLvs device Q17
i 17 pmos 0 5 3 3 " m 1 1 600e-9 w 6e-6 "
```

Normal Transmission Netlist

```
t 5 In input
t 4 Out output
n 0 /6
n 1 /5
n 2 /4
n 3 /3
n 4 /Out
n 5 /In
; pmos4 Instance /+3 = auLvs device Q0
d pmos D G S B (p D S)
i 0 pmos 4 0 2 2 " m 1 1 600e-9 w 24e-6 "
; pcapacitor Instance /+17 = auLvs device C1
d pcapacitor PLUS MINUS (p PLUS MINUS)
; pcapacitor Instance /+16 = auLvs device C2
; pcapacitor Instance /+15 = auLvs device C3
; pcapacitor Instance /+14 = auLvs device C4
; pcapacitor Instance /+13 = auLvs device C5
; pcapacitor Instance /+12 = auLvs device C6
; pcapacitor Instance /+11 = auLvs device C7
; pcapacitor Instance /+10 = auLvs device C8
; pcapacitor Instance /+9 = auLvs device C9
; pcapacitor Instance /+8 = auLvs device C10
; pcapacitor Instance /+7 = auLvs device C11
; pcapacitor Instance /+6 = auLvs device C12
; pcapacitor Instance /+5 = auLvs device C13
; pcapacitor Instance /+4 = auLvs device C14
; nmos4 Instance /+1 = auLvs device Q15
d nmos D G S B (p D S)
i 15 nmos 4 0 1 1 " m 1 1 600e-9 w 12e-6 "
; nmos4 Instance /+0 = auLvs device Q16
i 16 nmos 0 5 1 1 " m 1 1 600e-9 w 3e-6 "
; pmos4 Instance /+2 = auLvs device Q17
i 17 pmos 0 5 3 3 " m 1 1 600e-9 w 6e-6 "
```

Figure 4.27: netlists from Four Transmission Configurations

As one can see from Fig. 4.27, the four different configurations has no affect on the resulting netlists. This means that no matter what size the transmission line is, the netlists will always produce a matching result because there is no change in the parasitic capacitance. One hypothesis is that the netlist model of a circuit skips transmission line impact on a circuit. The netlist strictly deals with nets and even though one can see parasitic capacitance in the netlist, the amount of capacitance is not shown. Overlooking the amount of capacitance is a flaw in strictly verifying circuits based on the netlists. Another possible explanation for this is that the transmission lines were not long or wide enough to distinguish a change in the designs. The four configurations are at μm size and maybe having the lines run over a meter would detect a change in the netlists. Future work for this research could include why this phenomenon is occurring.

4.5 IP Cores

The next aspect in this research was to perform "trusting" of digital circuits taken from cores from opencores.org. The first step in this experiment was to choose cores taken from opencores.org. The goal behind this step was to choose differing cores that are applicable to the Air Force and have differing logic depth and gates. The first core chosen was a microprocessor without interlocked pipeline stages (MIPS) 16-bit processor. This is because it is a simple processor that would provide a proof of concept to the methodology behind this experiment. Figures 4.28, 4.29, and 4.30 show the top level Verilog code of a MIPS 16-bit processor.

```
 * Project: mips_16
 * Author: fxy
 * Description:
 *     top module of mips_16 cpu core. Technical details:
 *          1.  16-bit data width
 *          2.  classic 5-stage static pipeline, 1 branch delay slot, theoretical CPI is 1.0
 *          3.  pipeline is able to detect and prevent RAW hazards, no forwarding logic
 *          4.  8 general purpose register (reg 0 is special, according to mips architecture)
 *          5.  up to now supports 13 instructions, see ./doc/instruction_set.txt for details
 *
 * Revise history:
 *
 ***************************************************/
`timescale 1ns/1ps
`include "mips_16_defs.v"
module mips_16_core_top
(
    input                       clk,
    input                       rst,

    output  [`PC_WIDTH-1:0]     pc
);
    wire                        pipeline_stall_n ;
    wire    [5:0]               branch_offset_imm;
    wire                        branch_taken;
    wire    [15:0]              instruction;
    wire    [56:0]              ID_pipeline_reg_out;
    wire    [37:0]              EX_pipeline_reg_out;
    wire    [36:0]              MEM_pipeline_reg_out;

    wire    [2:0]               reg_read_addr_1;    // register file read port 1 address
    wire    [2:0]               reg_read_addr_2;    // register file read port 2 address
    wire    [15:0]              reg_read_data_1;    // register file read port 1 data
    wire    [15:0]              reg_read_data_2;    // register file read port 2 data
    wire    [2:0]               decoding_op_src1;       //source_1 register number
    wire    [2:0]               decoding_op_src2;       //source_2 register number
    wire    [2:0]               ex_op_dest;             //EX stage destinaton register number
    wire    [2:0]               mem_op_dest;            //MEM stage destinaton register number
    wire    [2:0]               wb_op_dest;             //WB stage destinaton register number
    wire                        reg_write_en;
    wire    [2:0]               reg_write_dest;
    wire    [15:0]              reg_write_data;

    IF_stage IF_stage_inst (
        .clk                    (clk),
        .rst                    (rst),
        .instruction_fetch_en   (pipeline_stall_n),
        .branch_offset_imm      (branch_offset_imm),
        .branch_taken           (branch_taken),
        .pc                     (pc),
        .instruction            (instruction)
    );
```

Figure 4.28: 16-bit MIPS Processor Top Level Module Part 1

```
ID_stage ID_stage_inst (
    .clk                    (clk),
    .rst                    (rst),
    .instruction_decode_en  (pipeline_stall_n),
    .pipeline_reg_out       (ID_pipeline_reg_out),
    .instruction            (instruction),
    .branch_offset_imm      (branch_offset_imm),
    .branch_taken           (branch_taken),
    .reg_read_addr_1        (reg_read_addr_1),  //
    .reg_read_addr_2        (reg_read_addr_2),  //
    .reg_read_data_1        (reg_read_data_1),  //
    .reg_read_data_2        (reg_read_data_2),  //
    .decoding_op_src1       (decoding_op_src1),
    .decoding_op_src2       (decoding_op_src2)
);

EX_stage EX_stage_inst (
    .clk                    (clk),
    .rst                    (rst),
    .pipeline_reg_in        (ID_pipeline_reg_out),
    .pipeline_reg_out       (EX_pipeline_reg_out),
    .ex_op_dest             (ex_op_dest)
);

MEM_stage MEM_stage_inst (
    .clk                    (clk),
    .rst                    (rst),
    .pipeline_reg_in        (EX_pipeline_reg_out),
    .pipeline_reg_out       (MEM_pipeline_reg_out),
    .mem_op_dest            (mem_op_dest)
);

WB_stage WB_stage_inst (
    .pipeline_reg_in        (MEM_pipeline_reg_out),
    .reg_write_en           (reg_write_en),
    .reg_write_dest         (reg_write_dest),
    .reg_write_data         (reg_write_data),
    .wb_op_dest             (wb_op_dest)
);

register_file register_file_inst (
    .clk                    (clk),
    .rst                    (rst),
    .reg_write_en           (reg_write_en),
    .reg_write_dest         (reg_write_dest),
    .reg_write_data         (reg_write_data),
    .reg_read_addr_1        (reg_read_addr_1),
    .reg_read_data_1        (reg_read_data_1),
    .reg_read_addr_2        (reg_read_addr_2),
    .reg_read_data_2        (reg_read_data_2)
);
```

Figure 4.29: 16-bit MIPS Processor Top Level Module Part 2

```
register_file register_file_inst (
    .clk                    (clk),
    .rst                    (rst),
    .reg_write_en           (reg_write_en),
    .reg_write_dest         (reg_write_dest),
    .reg_write_data         (reg_write_data),
    .reg_read_addr_1        (reg_read_addr_1),
    .reg_read_data_1        (reg_read_data_1),
    .reg_read_addr_2        (reg_read_addr_2),
    .reg_read_data_2        (reg_read_data_2)
);
```

Figure 4.30: 16-bit MIPS Processor Top Level Module Part 3

This is just the top-level module code, and there are many other modules underneath this module. Other cores that were selected for this experiment were: a cryptography core (e.g. AES), DSP core (e.g. FFT-based FIR filter), arithmetic core (e.g. accumulator), memory core (asynchronous SDRAM controller), and video controller core (H.264). These cores were selected because they are used in different applications and have differing logic depth and architecture.

The next step in this experiment was to synthesize the VHDL or verilog code in the RTL compiler tool in Cadence. The first core chosen was the accumulator core since it is the most basic out of all the cores listed above. The .tcl script used in this experiment is shown in Fig. 4.31.

The main goal of the .tcl file is to assign a library to use when synthesizing the code. The result of this experiment was unsuccessful. When trying to synthesize the VHDL code, the error message in Fig. 4.32 was shown.

Once this error was shown the library file was examined and that file is shown in Fig. 4.33. After doing many different changes to the .lib file and trying to use different libraries, it was determined that this experiment could not be done with the given libraries. Either the Cadence software at AFIT is not capable of synthesizing VHDL/Verilog code in the RTL compiler or the proper libraries have not been added to accomplish this task. The hypothesis taken for this experiment is the latter and that the proper libraries have not been added to the VLSI lab computers to accomplish this task.

Even though this experiment was a failure, the overall research has not been hindered as lessons were learned from this experiment and should be used in future work in continuing this area of research.

```
##rtl.tcl file adapted from http://ece.colorado.edu/~ecen5007/cadence/
##this tells the compiler where to look for the libraries

set_attribute lib_search_path /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta

## This defines the libraries to use

set_attribute library {cds.lib}

##This must point to your VHDL/verilog file
## CHANGE THIS LINE to your VHDL/verilog file name

read_hdl accu.v

## This buils the general block
elaborate

##this allows you to define a clock and the maximum allowable delays
## READ MORE ABOUT THIS SO THAT YOU CAN PROPERLY CREATE A TIMING FILE
set clock [define_clock -period 300 -name clk]
external delay -input 300 -edge rise clk
external delay -output 2000 -edge rise p1

##This synthesizes your code
synthesize -to_mapped

## This writes all your files
## change the tst to the name of your top level verilog
## CHANGE THIS LINE: CHANGE THE "accu" PART REMEMBER THIS
## FILENAME YOU WILL NEED IT WHEN SETTING UP THE PLACE & ROUTE
write -mapped > accu_synth.v

## THESE FILES ARE NOT REQUIRED, THE SDC FILE IS A TIMING FILE
write_script > script
```

Figure 4.31: .tcl Script used for RTL Compiler

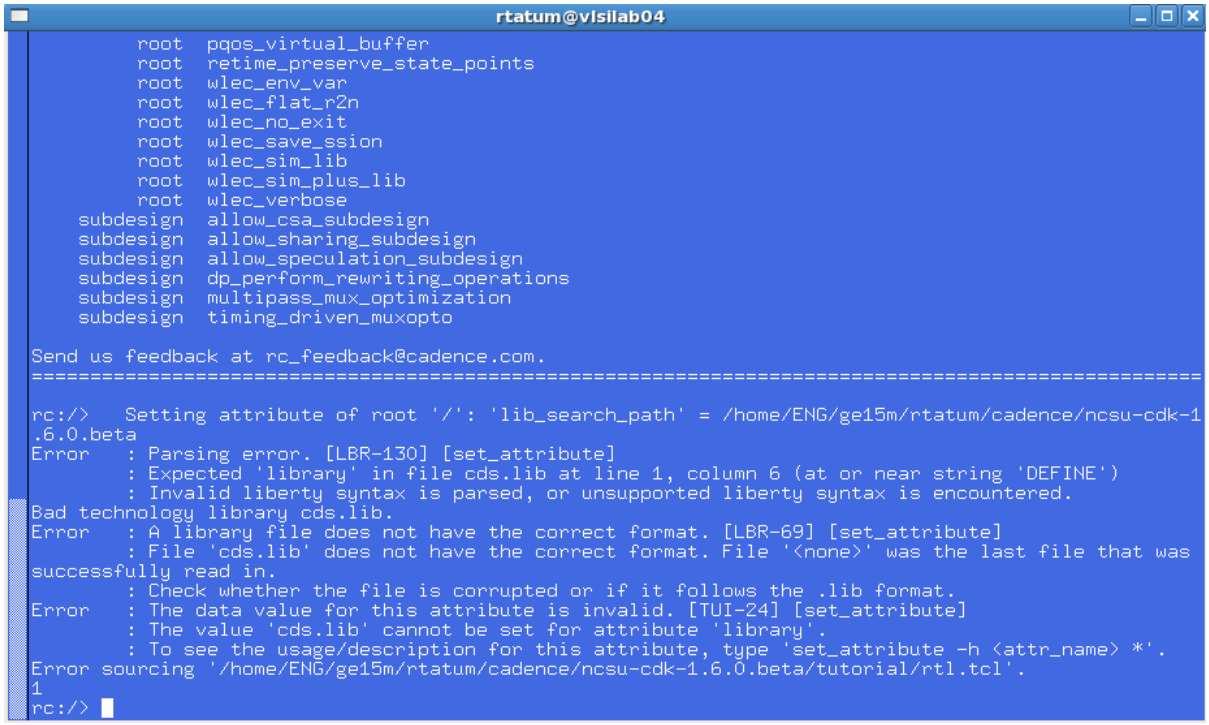

Figure 4.32: Error Message in RTL Compiler

```
DEFINE      basic                   $CDK_DIR/lib/basic
DEFINE      NCSU_Analog_Parts       $CDK_DIR/lib/NCSU_Analog_Parts
DEFINE      NCSU_Digital_Parts      $CDK_DIR/lib/NCSU_Digital_Parts
DEFINE      NCSU_TechLib_ami06      $CDK_DIR/lib/NCSU_TechLib_ami06
DEFINE      NCSU_TechLib_ami16      $CDK_DIR/lib/NCSU_TechLib_ami16
DEFINE      NCSU_TechLib_hp06       $CDK_DIR/lib/NCSU_TechLib_hp06
DEFINE      NCSU_TechLib_tsmc02     $CDK_DIR/lib/NCSU_TechLib_tsmc02
DEFINE      NCSU_TechLib_tsmc02d    $CDK_DIR/lib/NCSU_TechLib_tsmc02d
DEFINE      NCSU_TechLib_tsmc03     $CDK_DIR/lib/NCSU_TechLib_tsmc03
DEFINE      NCSU_TechLib_tsmc03d    $CDK_DIR/lib/NCSU_TechLib_tsmc03d
DEFINE      NCSU_TechLib_tsmc04_4M2P    $CDK_DIR/lib/NCSU_TechLib_tsmc04_4M2P
DEFINE EENG653 /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta/EENG653
DEFINE avTech /apps/vlsi/Cadence/ASSURA41/tools/assura/etc/avtech/avTech
#Removed by ddDeleteObj: DEFINE EENG653Tutorial /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta/EENG653Tutorial
DEFINE EENG653Tutorial /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta/EENG653Tutorial
DEFINE Midterm /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta/Midterm
DEFINE EENG695 /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta/EENG695
#Removed by ddDeleteObj: DEFINE UofU_Analog_Parts /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta/UofU_Analog_Parts/UofU_A
DEFINE UofU_Analog_Parts /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta//UofU_Analog_Parts
DEFINE UofU_Digital_v1_2 /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta//UofU_Digital_v1_2
DEFINE UofU_Example /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta//UofU_Example
DEFINE UofU_TechLib_ami06 /home/ENG/ge15m/rtatum/cadence/ncsu-cdk-1.6.0.beta//UofU_TechLib_ami06
```

Figure 4.33: .lib file used for RTL Compiler

4.5.1 Potential Solutions.

There are two solutions that the author has come up with to fix this issue. The first solution is to work with the head software engineer, Mr. David Doak, at AFIT and incorporate the correct libraries for this experiment to be conducted. Working with the head software engineer from AFIT will correctly identify if the problem was with the Linux operating system. If the issue was with the Linux operating system, Mr. Doak can help resolve this issue and eventually the IP core experiment can be conducted at the AFIT VLSI lab.

The second solution is to conduct the research at the AFRL MSDC as that lab has the capability to use this software with the proper libraries. Mr Christopher Taylor and Mr. Todd James at AFRL are very proficient in the Cadence Encounter software and working with these two individuals can help resolve the issue with the Cadence Encounter program. Once a proficiency is achieved in the Encounter software, the research can be conducted at AFIT's VLSI lab. From this experience, a necessary understanding of the Encounter software and Cadence libraries can aid in moving the research to AFIT.

4.6 Summary

The methodology outlined in Chapter 3 was applied to the different experiments and results were achieved. Basic transistor-level testing of three basic gates was conducted to see if they could be "trusted" under different circumstances. These circumstances were: custom vs. built-in library gates, parasitic capacitance inside the metal layout, and transmission lines. These experiments brought to light potential vulnerable areas when trusting integrated circuits. Testing of digital circuits was attempted and were unfortunately unsuccessful due to issues in the library provided by the Cadence software in the VLSI lab. Table 4.1 shows a quick summary on what has been done in the past, what I have done, and future work.

Table 4.1: Thesis Summary table

Previous Work	Performed Work	Future Work
Gate-level testing	Transistor-level testing	Testing on fabricated circuits

V. Conclusion and Future Work

5.1 Summary

This thesis has brought to light possible areas of vulnerabilities when "trusting" integrated circuits. It also attempted to take a look at differing digital IP cores to test the limitations of the DARPA TRUST program. Transistor-level testing was conducted to look at possible areas of exploitation. Custom and built-in circuits taken from the NCSU and University of Utah library were tested to see if having pre-made gate designs has any affect on the netlist matching process. It was determined that as long as the internal structure and pin names were matching between the schematic and metal layout, then the matching process was not affected. Parasitic capacitance was extracted from the metal layout version of gates to see if this had any affect on the netlist matching process because it is possible that parasitic capacitance can affect circuit operation over time. It was determined that the netlist matching rules file throws out parasitic capacitance during the matching process and therefore was not investigated. Transmission lines were examined between two inverters to see if the length and width of the line had any affect on the matching process. Since the rules file throws out parasitic capacitance, this test proved to result in matching netlists. Finally, digital IP cores were attempted to be tested to discover the limitations of the DARPA TRUST program and to ultimately try to characterize which gates are more likely to be tampered with. This experiment was unsuccessful due to the library provided to the AFIT VLSI lab. This research ultimately identified a few areas where "trusting" circuits could potentially lead to errors. It is suggested that the DARPA TRUST program start looking into doing transistor-level testing to try and prevent potential areas when matching circuits.

5.2 Future Work

This research only focused on doing transistor-level testing in the Cadence Virtuoso tool. Future work could potentially conduct transistor-level testing on a platform different from the Cadence Virtuoso tool.

5.2.1 IP Cores at AFIT.

The next step in this research would be to finish what was unable to be done in this thesis. Verification of IP cores using Cadence Encounter was the ultimate goal of one of the experiments. Future work could be importing a library so that testing and verification of digital IP cores can be conducted with the ultimate goal being characterizing digital gates based on vulnerabilities in being detected.

5.2.2 Parasitic Capacitance Potential Problem.

In this research parasitic capacitance was looked at to see if it had any affect on the matching process. However, the rules file provided in the LVS matching process threw out all of the parasitic capacitance components. Future work could be changing the rules file to not throw away the parasitic capacitance and then repeat the matching process. This could shed to light to see if there is some threshold of parasitic capacitance that would cause the netlists to not match.

5.2.3 Increasing Complexity in Custom vs. Built-in Circuits.

In this research the most complicated gate looked at was an XOR2 gate. Increasing the complexity to an adder or multiplier adds more nets and sees how scaling affects the custom vs. built-in matching process. Since there are many different ways to implement an adder, there are more ways for the built-in libraries to have differing internal architecture that the designer needs to be aware of. An example experiment is to test the many different adder implementations against one "control" adder implementation. Testing these different internal architectures against one custom gate will help draw conclusions if adding more nets could potentially change the matching process.

5.2.4 Fabrication.

The ultimate goal of this research is to test and verify fabricated circuits. Fabrication centers throw away inoperable circuits. Obtaining these circuits and conducting testing on them at the transistor-level can provide real world results and not just simulations. These circuits will have parasitic capacitance and transmission lines. Running a similar methodology outlined in this research with actual fabricated circuits can provide more insight on if parasitic capacitance and transmission lines affect a real world circuit.

Bibliography

[1] "Accumulators". URL http://www.csit-sun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/data/docs/xst/hdlcode7.html.

[2] "RTL Logic Synthesis Tutorial". URL http://www.siue.edu/~gengel/ece484LabMaterial/RTLsynthesisTut.pdf.

[3] "RTL Logic Synthesis Tutorial". URL http://eeweb.poly.edu/labs/nanovlsi/tutorials/soctutorials/Tutorial_RC.html.

[4] "C-130 Hercules", 2003. URL http://www.af.mil/AboutUs/FactSheets/Display/tabid/224/Article/104517/c-130-hercules.aspx.

[5] "DoD Microelectronics Strategic Management", 2005. URL http://www.pcfse.org/Meetings2005/minutesoct2005/glum.pps.

[6] "TRUST for Integrated Circuits", May 2006. URL https://www.fbo.gov/index?s=opportunity&mode=form&id=db4ea611cad3764814b6937fcab2180a&tab=core&_cview=1.

[7] "DARPA "TRUST in ICs" Effort", April 2007. URL http://www.dtic.mil/docs/citations/ADA503809.

[8] "DARPA "TRUST in ICs" Effort", April 2007. URL http://www.dtic.mil/docs/citations/ADA503809.

[9] "Dangerous Fakes.", October 2008. URL http://www.BusinessWeek.com/stories/2008-10-01/dangerous-fakes.

[10] "Counterfeits and the U.S. Industrial Base", 2010. URL http://www.semi.org/cms/groups/public/documents/webcontent/ctr038717.pdf.

[11] "DoD Software Assurance Initiative", 2010. URL https://acc.dau.mil/adl/en-US/25749/le/3178/DoD%20SW%20Assurance%20Initiative.pdf.

[12] "Trusted Manufacturing of Integrated Circuits for the Department of Defense", January 2010. URL http://www.ndia.org/Divisions/Divisions/Manufacturing/Documents/119A/5.

[13] "IN FOCUS: USAF receives last F-22 Raptor", May 2012. URL http://www.flightglobal.com/news/articles/in-focus-usaf-receives-last-f-22-raptor-371401/.

[14] "TRUSTED INTEGRATED CIRCUITS (TRUST)", April 2014. URL http://www.darpa.mil/Our_Work/MTO/Programs/Trusted_Integrated_Circuits_(TRUST).aspx.

[15] "System Design and Verification", 2015. URL http://www.cadence.com/products/sd/Pages/default.aspx.

[16] Alley, Charles L. *Electronic Engineering, 3rd Ed*. John Wiley & Sons, 1973.

[17] on Armed Services, Committee. *Inquiry into Counterfeit Electronic Parts in the Department of Defense Supply Chain*. Technical report, US Senate, 2012. URL http://www.armed-services.senate.gov/Publications/Counterfeit%20Electronic%20Parts.pdf.

[18] Beaumont, Bradley Hopkins, Mark and Tristan Newby. *A Survey of Hardware Trojan Taxonomy and Detection*. Technical report, University of Connecticut and Rice University, 2009.

[19] Behrens, Peter. "DoD Accredited Trusted Sources for Microelectronics". *National Semiconductor Corporation Trusted Solutions Business Unit*, 2010.

[20] Board, Defense Science. *Defense Industrial Base Assessment: Counterfeit Electronics*. Technical report, Office of Technology Evaluation Bureau of Industry & Security, 2010.

[21] of Commerce, Department. *Counterfeit Electronics Survey*. Technical report, Office of Technology Evaluation, 2009.

[22] of Defense, Deputy Secretary. *Defense Trusted Integrated Circuit Strategy*. Technical report, US Department of Defense, 2003.

[23] of Defense, Deputy Secretary. *Protection of Mission Critical Functions to Achieve Trusted Systems and Networks (TSN)*. Technical report, Office of Under Secretary of Defense for Acquisition, Technology, and Logistics, 2012.

[24] Lemnios, Zachary J. "Department of Defense Microelectronics Strategy". *Presentation, Trusted Sources, Supply Challenges and Solutions*, 2011.

[25] Mohammad Tehranipoor, Xuehui Zhang, Hassan Salmani. *Hardware Trojan Detection: Untrusted Manufactured Integrated Circuits*. Springer, 2013.

[26] Myers, Michael D. *Trust in Design for Integrated Circuits*. Technical report, Air Force Research Laboratory Mixed Signals Design Center, 2013.

[27] Sankman, Joseph. "Transitioning From Analog to Digital in Medical Designs", 2011. URL http://medsmagazine.com/2011/03/transitioning-from-analog-to-digital-in-medical-designs/.

[28] Seery, Michael. *COMPLEX VLSI FEATURE COMPARISON FOR COMMERCIAL MICROELECTRONICS VERIFICATION*. Master's thesis, Air Force Institute of Technology, 2013.

[29] Subhasish Mitra, H.-S. Philip Wong and Simon Wong. “Stopping Hardware Trojans in Their Tracks”. *IEEE Spectrum*, 2015.

[30] Williams, Brian Dupaix Todd James, Jason and Len Orlando. “TRUST in Design Tool Flow Overview”, 2012.

[31] Wynne, Michael. “Oral Presentation”, 2004.

[32] Zussa, Loic, Jean-Max Dutertre, Jessy Clediere, Bruno Robisson, and Assia Tria. “Investigation of timing constraints violation as a fault injection means”. *Departement Systemes et Architectures Securisees*, 2011.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2015	Master's Thesis	Oct 2013–Mar 2015

4. TITLE AND SUBTITLE

Exploration Of Digital Circuits And Transistor-Level Testing In The DARPA TRUST Program

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Tatum, Ralph K., Second Lieutenant, USAF

5d. PROJECT NUMBER

JON14G150

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-15-M-040

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Bradley Paul
Bradley.Paul@us.af.mil
(937) 528-8706
2241 Avionics Circle, Bldg 600
WPAFB, OH 43433-7318

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYDI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The need to verify correct circuit operation has grown in recent years due to adversaries ability to compromise DoD systems. The DARPA program addressed this issue and implemented the DARPA TRUST program to verify untrusted circuits using software. The DARPA TRUST program was initiated in 2006 and due to this the limitations and potential errors in the program have not yet been fully explored.

This research identifies the potential errors in the program by conducting transistor-level testing on circuits. The DARPA TRUST program currently operates at the gate-level and conducting various experiments at the transistor-level brought to light potential problems with current DARPA TRUST testing. The way that transistor-level verification is conducted is through netlist matching. A schematic of a circuit is created and the netlist is extracted, after that a metal layout of a circuit is created and the netlist is extracted. Once the two netlists are extracted, a matching program is used and the result determines if the verification process is successful. Parasitic capacitance was extracted in the metal layout version of a circuit and netlists were compared with the schematic version. Results show that parasitic capacitance is overlooked in the DARPA TRUST program even though this could potentially cause a fabricated device to fail. Transmission lines were simulated by creating metal wiring between two inverters. These metal lines mimic the operation of a transmission line. These transmission lines were experimented on and it was determined that the DARPA TRUST program does not effectively check for potential errors in transmission line fabrication. The results of this research brought to light the vulnerabilities in the DARPA TRUST program and addressed the need for the program to conduct transistor-level testing.

15. SUBJECT TERMS

DARPA TRUST, transistor-level testing, circuit verification, parasitic capacitance, transmission lines, microelectronics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Derrick Langley (ENG)
U	U	U	UU	87	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x6165 derrick.langley@afit.edu